บทที่ 3

การวิเคราะห์ ออกแบบระบบและการพัฒนาโปรแกรม

3.1 บทนำ

ในบทนี้จะกล่าวถึงขั้นตอนการวิเคราะห์ระบบ การออกแบบระบบและการพัฒนาโปรแกรมสำหรับระบบตรวจจับการบุกรุกเพื่อใช้ในงานวิจัยชิ้นนี้ โดยเริ่มจากภาพรวมของระบบตรวจจับการบุกรุกที่พัฒนาขึ้น วิธีการเก็บรวบรวมข้อมูลเข้า การวิเคราะห์ข้อมูลเข้า การวิเคราะห์การบุกรุก ถัดจากนั้นจะเป็นการอธิบายถึงการวิเคราะห์กฎที่ใช้สนับสนุนการตรวจจับการบุกรุกแต่ละข้อ ต่อด้วยการออกแบบข้อมูลที่จะใช้ในโปรแกรม การออกแบบโปรแกรมตามหลักเกณฑ์ของกฎแต่ละข้อ การออกแบบโปรแกรมโดยรวมทั้งระบบ และการพัฒนาโปรแกรม

3.2 ภาพรวมของระบบตรวจจับการบุกรุกที่พัฒนาขึ้น

ระบบตรวจจับการบุกรุกที่พัฒนาขึ้นมานี้เป็นเครื่องมือที่ผู้ดูแลระบบสามารถใช้ในการตรวจจับการบุกรุกที่เกิดขึ้นภภายในระบบ ตัวโปรแกรมตรวจจับการบุกรุกจะเริ่มทำงานเมื่อระบบหรือเครื่องที่ติดตั้งโปรแกรมถูกเปิดใช้งาน โดยจะคอยติดตามการทำงานของโปรเซสที่เกิดขึ้นทั้งหมดในขณะนั้นรวมทั้งโปรเซสลูกของโปรเซสเหล่านั้นด้วยเช่นกัน ลักษณะการทำงานของระบบตรวจจับการบุกรุกจะพิจารณาสถานะของโปรเซสในขณะนั้นหากพบว่าสถานะของโปรเซสอยู่ในสถานะสิทธิพิเศษตามคำนิยามที่กล่าวไว้ในบทที่ 2 ระบบตรวจจับการบุกรุกจะพิจารณาการเรียกใช้ system call ของ โปรเซสตาม กฎที่ใช้สนับสนุนการตรวจจับการบุกรุก หากพบว่าโปรเซสในขณะนั้นเข้าข่ายการบุกรุกนั่นคือเป็นไปตามกฎที่ใช้สนับสนุนข้อใดข้อหนึ่ง ระบบตรวจจับการบุกรุกจะหยุดการทำงานของโปรเซสนั้นรวมทั้งโปรเซสที่เกี่ยวข้องกับโปรเซสที่ถูกตรวจจับว่าเป็นการบุกรุกเพื่อป้องกันไม่ให้การบุกรุกนั้นกระทำได้สำเร็จหรือเพื่อป้องกันการกระทำที่จะเป็นผลมาจากการบุกรุกนั้น นอกจากนี้ระบบตรวจจับการบุกรุกจะทำการบันทึกข้อมูลการบุกรุกนี้ไว้ในแฟ้มบันทึกข้อมูลการบุกรุกด้วยเช่นกัน หัวข้อถัดไปจะกล่าวถึงการเก็บรวมรวมและการวิเคราะห์ข้อมูลเข้าที่ระบบตรวจจับใช้ในการพิจารณาการบุกรุก

3.3 การเก็บรวบรวมและการวิเคราะห์ข้อมูลเข้า

เนื่องจากงานวิจัยชิ้นนี้เน้นการตรวจจับการบุกรุกโดยพิจารณาจากการเรียกใช้ system call ของโปรแกรมหรือคำสั่งที่ถูกติดตาม ซึ่งนอกเหนือจากค่าที่เกี่ยวข้องกับ system call ที่ถูกเรียกใช้ในขณะนั้นอาทิเช่น ค่าของ system call ที่ถูกเรียกใช้ ค่าพารามิเตอร์ของ system call นั้น ค่าที่ถูกส่งกลับโดย system call ที่ถูกเรียกใช้ จุดที่ถูกติดตาม (trpoints) (ซึ่งจะอธิบายความหมายในย่อหน้าถัดไป) แล้วยังมีข้อมูลอื่นที่ใช้เป็นข้อมูลในการประกอบการพิจารณาการบุกรุกซึ่งได้แก่ ค่าประจำตัวผู้ใช้ (UID EUID GID EGID) เพื่อใช้พิจารณาสถานะของโปรเซสในขณะนั้น คำสั่งที่กระทำอยู่ในขณะนั้น รวมทั้งหมายเลขโปรเซสด้วย บนระบบปฏิบัติการยูนิกซ์ (NetBSD Unix Operating System) เราสามารถติดตามขั้นตอนการทำงานของโปรเซสต่าง ๆ ได้โดยใช้คำสั่ง "ktrace" ซึ่งการทำงานของคำสั่ง "ktrace" ดังกล่าวมีขั้นตอนการเรียกใช้ system call *ktrace()* ซึ่งเป็น system call ที่สำคัญที่ใช้ในการติดตามการเรียกใช้ system call ของโปรแกรมอื่น ๆ ในตารางที่ 3.1 แสดงผลของการติดตามการทำงานของคำสั่ง "ls" โดยใช้คำสั่ง "ktrace" ที่ติด มากับระบบปฏิบัติการยูนิกซ์ จะเห็นได้ว่าผลที่ได้จากการติดตามยังขาดข้อมูลที่เกี่ยวกับค่าประจำตัวผู้ใช้ซึ่งเป็นข้อมูลสำคัญในการพิจารณาสถานะของโปรเซส ฉะนั้นเพื่อให้ได้ข้อมูลค่าประจำตัวผู้ใช้ ผู้วิจัยได้ทำการแก้ไข system call *ktrace()* เพื่อให้แสดงค่าของ UID EUID GID และ EGID ในตารางที่ 3.2 และ 3.4 เป็นผลการติดตามที่ได้จากการใช้คำสั่ง "ktrace" ที่ได้มีการแก้ไข system call *ktrace()* เพื่อให้แสดงค่าประจำตัวผู้ใช้แล้วจะเห็นได้ว่าเราสามารถพิจารณาสถานะของโปรเซสได้จากค่าประจำตัวผู้ใช้เหล่านี้ สำหรับการวิเคราะห์ผลที่ได้จากการติดตามการทำงานของคำสั่งเราสามารถแบ่งกลุ่มของคำสั่งที่ถูกติดตามออกเป็นสองกลุ่มใหญ่ คือกลุ่มคำสั่งปกติและกลุ่มคำสั่งพิเศษซึ่งถูกอธิบายไว้ในหัวข้อที่ 3.3.1 และ 3.3.2 ส่วนหนึ่งของผลจากการติดตามการทำงานของคำสั่งทั้งสองกลุ่มถูกแสดงไว้ในตารางที่ 3.2 และ ตารางที่ 3.4

ตารางที่ 3.1 ส่วนหนึ่งของผลที่ได้จากการติดตามการเรียกใช้ system call ของคำสั่ง "ls" โดยใช้คำสั่ง *ktrace*

บรรทัด	PID	cmd	trpoints	syscall(parameter / return value)
1	...			
2	"l"			
3	2023	ksh	RET	write 1
4	2023	ksh	CALL	read(0,0xbfbff237,0x1)
5	2023	ksh	GIO	fd 0 read 1 bytes
6	"s"			
7	2023	ksh	RET	read 1
8	2023	ksh	CALL	write(0x2,0x8074290,0x1)
9	2023	ksh	GIO	fd 2 wrote 1 bytes
10	...			
11	2023	ksh	RET	read 1
12	2023	ksh	CALL	write(0x2,0x8074290,0x2)
13	2023	ksh	GIO	fd 2 wrote 2 bytes
14	"\r"			
15	....			
16	2023	ksh	NAMI	"/bin/ls"
17	2023	ksh	RET	__stat13 0
18	2023	ksh	CALL	access(0x807d0c8,0x1)
19	2023	ksh	NAMI	"/bin/ls"
20	2023	ksh	RET	access 0
21	2023	ksh	NAMI	"/bin/ls"
22	2023	ksh	RET	__stat13 0
23	2023	ksh	CALL	access(0x807d0c8,0x1)
24	2023	ksh	NAMI	"/bin/ls"
25	2023	ksh	RET	access 0
26	2023	ksh	CALL	fork
27	5591	ksh	EMUL	"netbsd"
28	2023	ksh	RET	fork 5591/0x15d7
30	5591	ksh	CALL	execve(0x8079408,0x807d024,0x807e020)
31	5591	ksh	NAMI	"/bin/ls"
32	5591	ksh	NAMI	"/libexec/ld.elf_so"
33	5591	ls	CALL	mmap(0,0x8000,0x3,0x1002,0xffffffff,0,0,0)
35	5591	ls	CALL	exit(0)
36	2023	ksh	PSIG	SIGCHLD caught handler=0x8064a28 mask=(20) code=0x0
37	2023	ksh	CALL	getrusage(0,0xbfbff3c0)

ตารางที่ 3.2 ส่วนหนึ่งของผลที่ได้จากการติดตามการเรียกใช้ system call ของคำสั่งปกติโดยใช้คำสั่ง *ktrace* ที่แก้ไขแล้ว

บรรทัด	UID	EUID	GID	EGID	PID	cmd	trpoints	syscall(parameter / return value)
0	ส่วนที่ 1							
1	...							
2	"l"							
3	1000	1000	100	100	2023	ksh	RET	write 1
4	1000	1000	100	100	2023	ksh	CALL	read(0,0xbfbff237,0x1)
5	1000	1000	100	100	2023	ksh	GIO	fd 0 read 1 bytes
6	"s"							
7	1000	1000	100	100	2023	ksh	RET	read 1
8	1000	1000	100	100	2023	ksh	CALL	write(0x2,0x8074290,0x1)
9	1000	1000	100	100	2023	ksh	GIO	fd 2 wrote 1 bytes
10	...							
11	1000	1000	100	100	2023	ksh	NAMI	"/bin/ls"
12	1000	1000	100	100	2023	ksh	RET	__stat13 0
13	1000	1000	100	100	2023	ksh	CALL	access(0x807d0c8,0x1)
14	1000	1000	100	100	2023	ksh	NAMI	"/bin/ls"
15	1000	1000	100	100	2023	ksh	RET	access 0
16	ส่วนที่ 2							
17	1000	1000	100	100	2023	ksh	NAMI	"/bin/ls"
18	1000	1000	100	100	2023	ksh	RET	__stat13 0
19	1000	1000	100	100	2023	ksh	CALL	access(0x807d0c8,0x1)
20	1000	1000	100	100	2023	ksh	NAMI	"/bin/ls"
21	1000	1000	100	100	2023	ksh	RET	access 0
22	....							
23	ส่วนที่ 3							
24	1000	1000	100	100	2023	ksh	CALL	fork
25	1000	1000	100	100	5591	ksh	EMUL	"netbsd"
26	1000	1000	100	100	2023	ksh	RET	fork 5591/0x15d7
27	.....							
28	ส่วนที่ 4							
29	1000	1000	100	100	5591	ksh	CALL	execve(0x8079408,0x807d024,
30							0x807e020)	
31	1000	1000	100	100	5591	ksh	NAMI	"/bin/ls"
32	1000	1000	100	100	5591	ksh	NAMI	"/libexec/ld.elf_so"
33	1000	1000	100	100	5591	ls	EMUL	"netbsd"
34	1000	1000	100	100	5591	ls	RET	execve JUSTRETURN
35	1000	1000	100	100	5591	ls	CALL	
36							mmap(0,0x8000,0x3,0x1002,0xffffffff,0,0,0)	
37	.....							
38	1000	1000	100	100	5591	ls	CALL	exit(0)
39	ส่วนที่ 5							
40	1000	1000	100	100	2023	ksh	PSIG	SIGCHLD caught
41							handler=0x8064a28 mask=(20) code=0x0	
42	1000	1000	100	100	2023	ksh	CALL	getrusage(0,0xbfbff3c0)

จากตารางที่ 3.1 และ 3.2 ซึ่งแสดงให้เห็นถึงผลจากการติดตามการทำงานของคำสั่ง สามารถแยกอธิบายความหมายในส่วนของ trpoints ได้ดังตารางที่ 3.3

ตารางที่ 3.3 trpoints(trace points) ของ system call *ktrace()*

Trace points	**Definition**
CALL	ติดตาม system call
RET	ติดตามค่าที่ส่งกลับจาก system call
NAMEI	ติดตามเส้นทางของแฟ้มที่ถูกเรียกใช้
GENIO	ติดตาม I/O ทั้งหมด
PSIG	ติดตาม posted signal
CSW	ติดตาม context switch points
INHERIT	ติดตามโปรเซสลูกที่จะเกิดขึ้น

3.3.1 ผลการวิเคราะห์ที่ได้จากคำสั่งปกติ

คำสั่งปกติในที่นี้หมายถึงคำสั่งที่ไม่มีการเปลี่ยนแปลงค่าของ UID EUID GID หรือ EGID ของโปรเซสในขณะใดขณะหนึ่งตั้งแต่คำสั่งถูกเรียกใช้งานจนจบการทำงานของคำสั่งนั้น นั่นคือเป็นคำสั่งที่ไม่ก่อให้เกิดการเปลี่ยนแปลงสถานะของโปรเซสตั้งแต่เริ่มทำงานจนจบกระบวนการ ตัวอย่างได้แก่ คำสั่ง *ls* ดังแสดงในตารางที่ 3.3 ข้อมูลที่ได้ในแต่ละบรรทัดสามารถแบ่งออกได้เป็น 2 กลุ่มคือ ส่วนที่หนึ่งเป็นค่าที่เกี่ยวข้องกับตัวผู้ใช้ซึ่งประกอบด้วยค่าประจำตัวผู้ใช้ (UID EUID GID EGID) และหมายเลขโปรเซส (PID) ส่วนที่สองเป็นค่าที่เกี่ยวข้องกับ system call ได้แก่ คำสั่งที่เรียกใช้ system call (command) รายการที่ต้องการติดตาม (trpoints) พารามิเตอร์หรือค่าที่ถูกส่งกลับ (parameter / return value) และเพื่อให้เกิดความเข้าใจมากขึ้นตัวเลขแสดงบรรทัดจะถูกนำมาใช้ในการอ้างอิงตำแหน่ง ข้อมูลที่ได้ทั้งหมดสามารถแยกอธิบายได้เป็น 5 ส่วนตามลำดับ คือในส่วนที่ 1 เริ่มจากคำสั่ง "ksh" รอรับการทำงานจากผู้ใช้เมื่อผู้ใช้พิมพ์คำสั่ง "ls" คำสั่งจะถูกอ่านเข้ามาครั้งละตัวอักษรและถูกแสดงออกมาดังตัวอย่างในบรรทัดที่ 4 เมื่อคำสั่งถูกอ่านจนจบระบบจะทำการตรวจสอบว่าคำสั่งที่ผู้ใช้พิมพ์เข้ามานั้นมีอยู่ในระบบหรือไม่ โดยจะเริ่มค้นหาจากไดเรกทอรีที่ถูกระบุไว้ในตัวแปร PATH ตามลำดับดังแสดงในส่วนที่ 2 ถ้าคำสั่งที่ผู้ใช้ระบุมีอยู่จริงซึ่งใน

ที่นี่คือคำสั่ง "ls" คำสั่ง ksh ซึ่งเป็นคำสั่งที่ทำงานอยู่ในขณะนั้นจะแตกโปรเซสใหม่โดยการเรียก system call *fork()* (บรรทัดที่ 24) ซึ่งอยู่ในส่วนที่ 3 ทำให้ได้หมายเลขโปรเซสใหม่ (บรรทัดที่ 25) เป็นโปรเซสลูกของโปรเซสที่ทำคำสั่ง "ksh" ถัดมาตัว โปรเซสลูกจะเรียก system call *execve()* เพื่อรันคำสั่ง "ls" จะเห็นได้จากส่วนที่ 4 (บรรทัดที่ 34) สุดท้ายในส่วนที่ 5 เมื่อจบสิ้นกระบวนการทำงานของคำสั่ง "ls" ซึ่งเป็นโปรเซสลูกก็จะกลับเข้าสู่กระบวนการทำงานของโปรเซสแม่ซึ่งเป็นกระบวนทำงานของคำสั่ง "ksh" รอคำสั่ง (บรรทัดที่ 45) จะเห็นได้ว่าตลอดการติดตามการทำงานของคำสั่ง "ls" ค่าของ UID EUID GID และ EGID ของโปรเซสในกระบวนการทำงานตั้งแต่เริ่มคำสั่ง "ls" จนกระทั่งจบกระบวนการ จากตัวอย่างในที่นี้คือ 1000 1000 100 และ 100 ไม่มีการเปลี่ยนแปลง

3.3.2 ผลการวิเคราะห์ที่ได้จากคำสั่งพิเศษ

คำสั่งพิเศษในที่นี้หมายถึงคำสั่งที่ต้องใช้สิทธิเทียบเท่าผู้ใช้ที่มีสิทธิในการเรียกใช้หรือเจ้าของทรัพยากรที่ถูกเรียกใช้ในขณะนั้น หรือมีสิทธิเทียบเท่าผู้ใช้ที่มีสิทธิสูงสุดหรือกลุ่มระบบโดยทั่วไปแล้วคำสั่งพิเศษพวกนี้จะจัดอยู่ในคำสั่งประเภท setuid โปรแกรม คือคำสั่งที่มีการกำหนดค่าของ set-user-id นั่นคือในขณะที่คำสั่งนั้นทำงานค่า EUID ของโปรเซสจะถูกเปลี่ยนให้มีค่าเท่ากับเจ้าของคำสั่งนั้นเพื่อให้มีสิทธิในการเรียกใช้ทรัพยากรที่ต้องใช้สิทธิของเจ้าของคำสั่ง ตัวอย่างเช่น ถ้าเจ้าของแฟ้มเป็นผู้ใช้ที่มีสิทธิสูงสุด และ set-user-bit ของแฟ้มนั้นถูกกำหนดดังนั้นขณะที่แฟ้มโปรแกรมนี้เข้าสู่กระบวนการทำงานโปรเซสก็จะได้รับสิทธิของผู้ใช้สูงสุด[34] ตัวอย่างคำสั่งประเภท setuid โปรแกรมได้แก่คำสั่ง "passwd" ซึ่งใช้ในการเปลี่ยนรหัสผ่านคำสั่งนี้มีการกำหนด set-user-id bit โดยมีเจ้าของแฟ้มเป็นผู้ใช้ที่มีสิทธิสูงสุด พิจารณาสมบัติของคำสั่ง "passwd"

```
-r-sr-xr-x 1 root root 13476 Aug 7 2001 /usr/bin/passwd
```

ตัวอักษร "s" ในลำดับที่ 4 แสดงให้เห็นว่าคำสั่ง "passwd" มีการกำหนด set-user-id bitโดยเจ้าของคำสั่งนี้คือ root หรือผู้ใช้ที่มีสิทธิสูงสุด ดังนั้นในกระบวนการทำงานโปรเซสจะมีการเปลี่ยนแปลงค่าของ EUID ให้มีค่าเท่ากับ 0 ซึ่งเป็นค่าเดียวกับค่า UID ของผู้ใช้ที่มีสิทธิสูงสุดซึ่งเป็นเจ้าของแฟ้มเพื่อให้สามารถมีสิทธิในการเขียนแฟ้ม /etc/passwd ได้ซึ่งผู้ใช้โ ดยทั่วไปจะมีสิทธิเพียงแค่อ่านแฟ้มเท่านั้นแต่จะไม่มีสิทธิในการแก้ไขข้อมูลในแฟ้ม ตารางที่ 3.4 แสดงตัวอย่างข้อมูลที่ได้จากการติดตามการทำงานของคำสั่ง "passwd" ด้วยคำสั่ง "ktrace" ข้อมูลที่แสดงในตารางแต่ละบรรทัดแบ่งเป็นส่วนเช่นเดียวกับการแสดงผลในตารางที่ 3.3 ในส่วนของการวิเคราะห์ข้อมูล

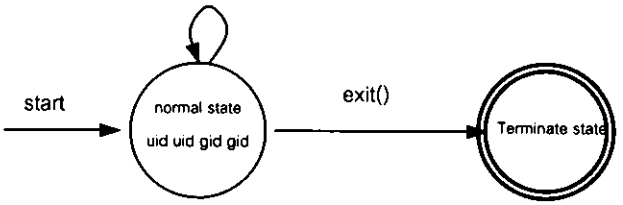
ข้อแตกต่างที่เห็นได้ชัดก็คือจากขั้นตอนการทำงานซึ่งเป็นการทำงานของคำสั่ง "passwd" เองจะเห็นได้ว่าโปรเซสมีการเปลี่ยนแปลงของค่า EUID (บรรทัดที่ 17) จากเดิมซึ่งมีค่าตัวเลขเท่ากับ 100 ซึ่งเป็นค่าประจำตัวของผู้ใช้ที่เรียกใช้คำสั่ง *passwd* เปลี่ยนมาเป็นตัวเลข 0 ซึ่งเป็นค่าประจำตัวของผู้ใช้ที่มีสิทธิสูงสุดเพื่อให้สามารถเรียกใช้ทรัพยากรที่จำเป็นในการเปลี่ยนรหัสผ่านได้ หลังจากนั้นเมื่อจบกระบวนการการทำงานของคำสั่ง *passwd* นั่นคือรหัสผ่านถูกเปลี่ยนเรียบร้อยแล้วโปรเซสแม่ซึ่งคือกระบวนการทำงานของคำสั่ง "ksh" รอรับคำสั่งจากผู้ใช้ต่อไป (บรรทัดที่ 33)

ตารางที่ 3.4 ส่วนหนึ่งของผลที่ได้จากการติดตามการเรียกใช้ system call ของคำสั่งพิเศษโดยใช้คำสั่ง *ktrace* ที่แก้ไขแล้ว

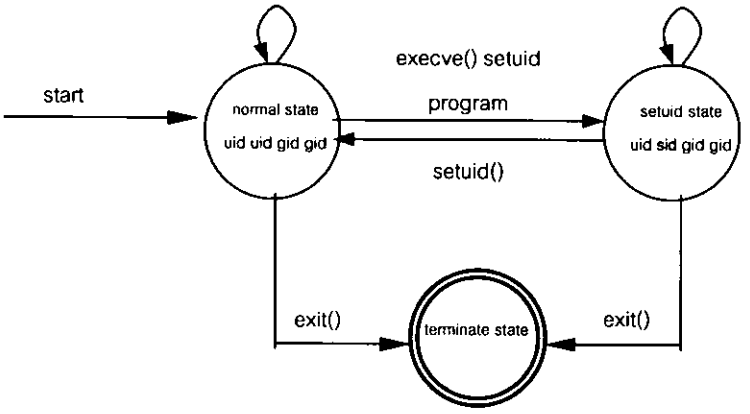
บรรทัด	UID	EUID	GID	EGID	PID	cmd	trpoints	syscall(parameter / return value)
0	...							
1	ส่วนที่ 2							
2	1000	1000	100	100	4520	ksh	CALL	access(0x807d0c8,0x1)
3	1000	1000	100	100	4520	ksh	NAMI	"/usr/bin/passwd"
4	1000	1000	100	100	4520	ksh	RET	access 0
5	...							
6	ส่วนที่ 3							
7	1000	1000	100	100	4520	ksh	CALL	fork
8	1000	1000	100	100	10873	ksh	EMUL	"netbsd"
9	1000	1000	100	100	4520	ksh	RET	fork 10873/0x2a79
10	1000	1000	100	100	10873	ksh	RET	fork 0
11	...							
12	ส่วนที่ 4							
13	1000	1000	100	100	10873	ksh	CALL	execve
14						(0x807d0c8,0x807d024,0x807e020)		
15	1000	1000	100	100	10873	ksh	NAMI	"/usr/bin/passwd"
16	1000	1000	100	100	10873	ksh	NAMI	"/usr/libexec/ld.elf_so"
17	1000	**0**	100	100	10873	passwd	EMUL	"netbsd"
18	1000	**0**	100	100	10873	passwd	RET	execve JUSTRETURN
19	....							
20	1000	**0**	100	100	10873	passwd	CALL	open(0x480708c0,0xa01,0x180
21	1000	**0**	100	100	10873	passwd	NAMI	"/etc/ptmp"
22	1000	**0**	100	100	10873	passwd	RET	open 3
23	1000	**0**	100	100	10873	passwd	CALL	umask(0x12)
24	1000	**0**	100	100	10873	passwd	RET	umask 0
25	1000	**0**	100	100	10873	passwd	CALL	open(0x804b0d0,0,0)
26	1000	**0**	100	100	10873	passwd	NAMI	"/etc/master.passwd"
27	....							
28	ส่วนที่ 5							
29	1000	**0**	100	100	10873	passwd	CALL	exit(0)
30	...							
31	1000	1000	100	100	4520	ksh	CALL	write(0x2,0x8074290,0x2)
32	1000	1000	100	100	4520	ksh	GIO	fd 2 wrote 2 bytes
33	"$ "							

3.3.3 สรุปผลการวิเคราะห์การติดตามกระบวนการทำงานของคำสั่งบนระบบปฏิบัติการยูนิกซ์

จากข้อมูลข้างต้นในหัวข้อ 3.3.1 และ 3.3.2 ข้อแตกต่างที่เห็นได้ชัดเจนระหว่างกระบวนการทำงานของคำสั่งปกติและคำสั่งพิเศษก็คือในกรณีของคำสั่งพิเศษจะมีช่วงเวลาขณะหนึ่งที่ค่า UID EUID GID หรือ EGID เกิดการเปลี่ยนแปลงทำให้สถานะของโปรเซสเปลี่ยนจากสถานะปกติเป็นสถานะพิเศษนั่นคือค่า EUID เปลี่ยนเป็น 0 ซึ่งสามารถใช้ Finite State Machine (FSM) และคำนิยามสถานะที่กำหนดไว้ในบทที่ 2 อธิบายได้ดังนี้



ภาพประกอบที่ 3.1 State transition diagram แสดงการทำงานของคำสั่งปกติ



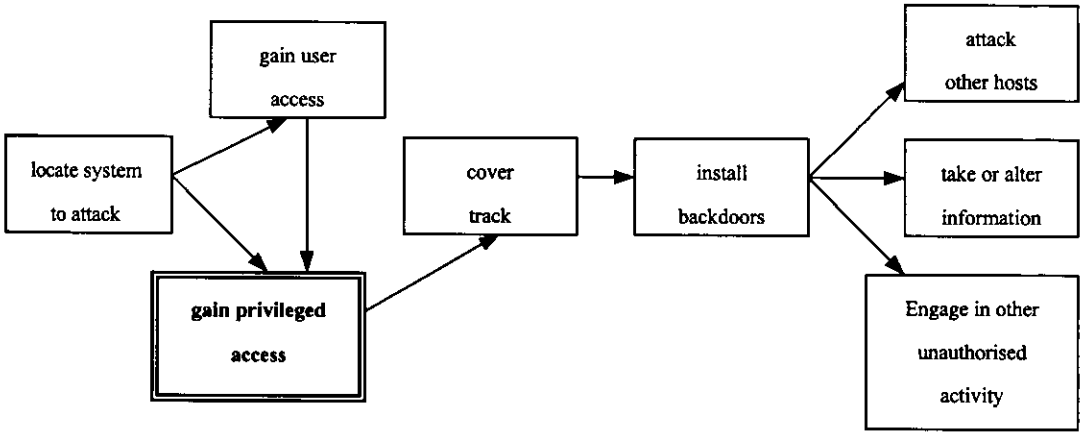
ภาพประกอบที่ 3.2 State transition diagram แสดงการทำงานของคำสั่งพิเศษ (setuid โปรแกรม)

จากภาพประกอบที่ 3.1 จะเห็นได้ชัดว่าในกระบวนการทำงานของคำสั่งปกติสถานะของโปรเซสตั้งแต่เริ่มต้นกระบวนการทำงานไม่มีเปลี่ยนแปลงจนกระทั่งสิ้นสุดการทำงาน และในภาพประกอบที่ 3.2 ซึ่งแสดงไดอะแกรมการเปลี่ยนแปลงสถานะในกระบวนการทำงานของคำสั่งพิเศษในที่นี้เป็นคำสั่งประเภท setuid โปรแกรมซึ่งจะมีการเปลี่ยนแปลงค่าของ euid เป็น 0 และเมื่อสิ้นสุดกระบวนการที่ต้องใช้สิทธิพิเศษค่า euid จะกลับมาเป็นค่า euid เดิมเหมือนในสถานะปกติก่อนจะสิ้นสุดการทำงานของคำสั่งด้วย system call *exit()* หรืออาจจะเรียก system call *exit()* เพื่อสิ้นสุดการทำงานของคำสั่งนั้นโดยไม่มีการเรียก system call *setuid()* เพื่อเปลี่ยนค่า euid กลับไปยังสถานะเดิม ในกรณีนี้ตัวระบบจะทำการตัดสิทธิของผู้ใช้ให้กลับมามีสิทธิเป็นปกติหลังจากที่สิ้นสุดการทำงานของคำสั่งนั้นหรือหลังจากที่ system call *exit()* ถูกเรียกใช้โดยอัตโนมัติ

ในการพัฒนาระบบตรวจจับการบุกรุกสำหรับงานวิจัยชิ้นนี้ ค่าที่ถูกนำไปใช้ได้แก่ค่า UID, EUID, GID และ EGID รวมทั้งค่า PID, cmd, trpoints, system call parameters และ return value ค่าเหล่านี้จะถูกอธิบายไว้ในหัวข้อ 3.6 การออกแบบข้อมูลนำเข้า

3.4 การวิเคราะห์การบุกรุก

จากข้อมูลตารางที่ 2.1 – 2.3 บทที่2 ในเรื่องของการบุกรุกจะเห็นได้ว่าโดยส่วนใหญ่จุดอ่อนของระบบซึ่งเป็นเป้าหมายของผู้บุกรุกจะเป็นโปรแกรมประเภท setuid หรือ setgid โปรแกรม ซึ่งทำงานด้วยสิทธิของผู้ใช้ที่มีสิทธิสูงสุด เนื่องจากในขณะที่โปรแกรมเหล่านี้ทำงานจะมีการเปลี่ยนแปลงค่าของ EUID ให้มีค่าเท่ากับ 0 ทำให้มีสิทธิเพียงพอที่จะใช้ทรัพยากรในระบบได้ นอกจากนี้จากภาพประกอบที่ 3.3 ซึ่งเป็นบทความจาก CERT/CC Overview Incident and Vulnerability Trends แสดงให้เห็นถึงรูปแบบการโจมตีของผู้บุกรุกแสดงให้เห็นว่าเมื่อผู้บุกรุกสามารถเข้ามาในระบบได้แล้วขั้นตอนก่อนที่จะทำให้ผู้บุกรุกสามารถควบคุมการทำงานของระบบได้ นั่นคือการได้มาซึ่งสิทธิพิเศษเช่นเดียวกันและสิทธิที่จะทำให้ผู้บุกรุกสามารถจะควบคุมทรัพยากรในระบบได้ซึ่งหมายถึงสิทธิของผู้ใช้สูงสุด จะเห็นได้อย่างชัดเจนว่าผู้บุกรุกสามารถใช้ข้อบกพร่องของโปรแกรมประเภท setuid โปรแกรมหรือ setgid โปรแกรมเป็นจุดอ่อนในการบุกรุกระบบได้สำเร็จ ในปัจจุบันข้อมูลเกี่ยวกับจุดอ่อนเหล่านี้มีเผยแพร่ทั่วไปบนอินเตอร์เน็ต ถ้าหากผู้ดูแลระบบละเลยในการแก้ไขจุดบกพร่องเหล่านี้ ก็จะเป็นการง่ายสำหรับผู้บุกรุกที่จะโจมตีระบบ



ภาพประกอบที่ 3.3 รูปแบบการโจมตีโดยทั่วไป [37]

ฉะนั้นงานวิทยานิพนธ์ชิ้นนี้จึงสนใจในกรณีที่มีการเรียกใช้โปรแกรมประเภทsetuid โปรแกรมหรือ setgid โปรแกรมที่ทำงานโดยใช้สิทธิของผู้ใช้สูงสุดเนื่องจากเป็นข้อสังเกตแรกที่จะสามารถวิเคราะห์ได้ว่าพฤติกรรมการใช้งานที่เกิดขึ้นต่อเนื่องเมื่อโปรเซสอยู่ในสถานะสิทธิพิเศษจากการเรียกใช้งานโปรแกรมเหล่านั้นเป็นพฤติกรรมที่เข้าข่ายการบุกรุกหรือไม่ โดยนำข้อมูลซึ่งเป็นผลกระทบจากการบุกรุกมาวิเคราะห์ร่วมทำให้ได้กฎที่ใช้สนับสนุนการตรวจจับพฤติกรรมเหล่านั้น นั่นคือในกรณีของผลกระทบที่เกิดจากการที่ผู้บุกรุกเรียกใช้โปรแกรมอื่นในขณะที่อยู่ในสถานะเทียบเท่าผู้ใช้หรือกลุ่มผู้ใช้ที่มีสิทธิสูงสุดหรือในขณะที่โปรเซสขณะนั้นมีค่า EUID หรือ EGID เท่ากับ 0 สามารถตรวจจับด้วยกฎข้อที่ 1 หรือในกรณีที่ผู้บุกรุกเปลี่ยนสถานะให้กลายเป็นผู้ใช้ที่มีสิทธิสูงสุดจริง ๆ นั่นคือมีการเปลี่ยนแปลงค่า UID หรือ GID เป็น 0 ในขณะที่โปรเซสขณะนั้นมีค่า EUID หรือ EGID เท่ากับ 0 สามารถตรวจจับด้วยกฎข้อที่ 0 กรณีที่ผู้บุกรุกแก้ไขแฟ้มระบบสามารถตรวจจับด้วยกฎข้อที่ 3 ในส่วนของผลกระทบอื่น ๆ อย่างเช่นผู้บุกรุกติดตั้งโปรแกรมม้าโทรจัน หรือสร้างแฟ้มประเภท setuid หรือ setgid สามารถตรวจจับด้วยกฎข้อที่ 2 และในกรณีที่ผู้บุกรุกเรียกใช้ system call บางตัวเช่น *mount()* ตรวจจับได้ด้วยกฎข้อที่ 5 รวมทั้งกรณีที่ผู้บุกรุกสร้างบัญชีรายชื่อผู้ใช้ใหม่ตรวจจับได้ด้วยกฎข้อที่ 4 ในหัวข้อถัดไปจะกล่าวถึงรายละเอียดในการวิเคราะห์กฎที่ใช้สนับสนุนการตรวจจับการบุกรุก

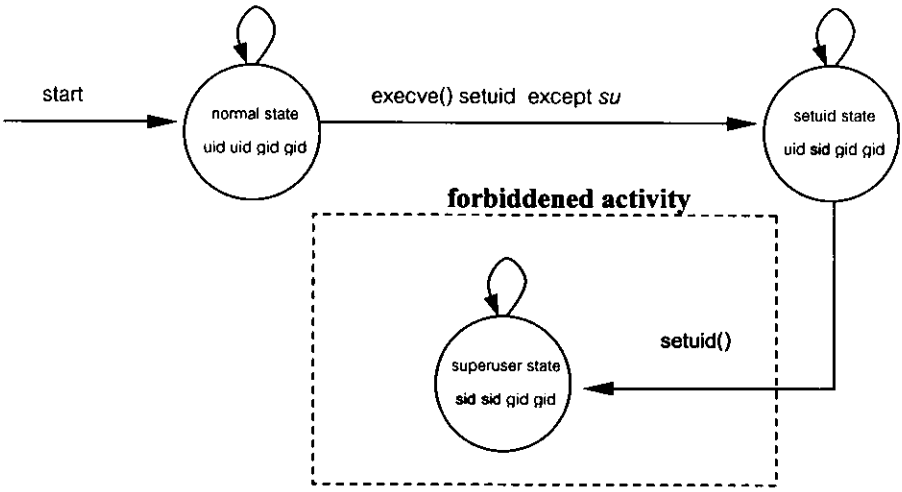
3.5 การวิเคราะห์กฎที่ใช้สนับสนุนการตรวจจับการบุกรุก

เนื่องจากประเด็นหลักในการตรวจจับการบุกรุกจะพิจารณาจากการเปลี่ยนสถานะของโปรเซสจากสถานะปกติเป็นสถานะพิเศษเพื่อให้ง่ายในการวิเคราะห์กฎแต่ละข้อจะใช้ finite state machine เป็นเครื่องมือในการอธิบาย (ดังแสดงในภาพประกอบที่ 3.4 – 3.12)

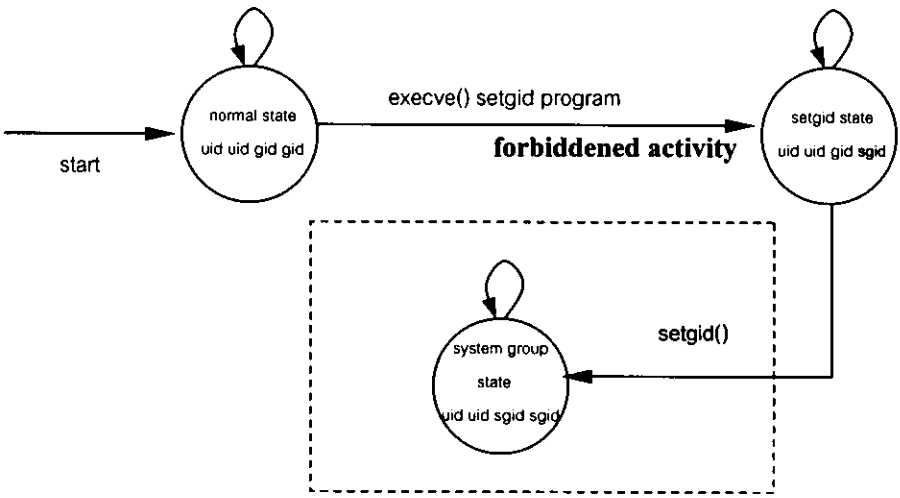
3.5.1 กฎข้อที่ 0 เมื่อโปรเซสอยู่ในสถานะสิทธิพิเศษอนุญาตให้ system call *setreuid()* และ *setregid()* เท่านั้นที่สามารถเปลี่ยน UID หรือ GID ได้

ผลกระทบที่เกิดขึ้น ถ้าหากผู้บุกรุกสามารถเรียกใช้ system call *setreuid()* หรือ *setregid()* ได้สำเร็จในขณะที่โปรเซสอยู่ในสถานะที่มีค่า EUID หรือ EGID เป็น 0 จะทำให้ผู้บุกรุกกลายเป็นผู้ใช้ที่มีสิทธิสูงสุดหรือผู้ใช้ที่อยู่ในกลุ่มสูงสุดเนื่องจากการเรียกใช้ system call 2 ตัวนี้จะเปลี่ยนแปลงค่า UID หรือ GID มีค่าเทากับ 0

การตรวจจับ ถ้ามีการใช้ system call ตัวอื่นหรือโปรแกรมอื่นเพื่อทำการเปลี่ยนแปลงค่า UID หรือ GID ให้ถือว่าเข้าข่ายการบุกรุก จากภาพประกอบที่ 3.4 กระบวนการทำงานเริ่มต้นจากสถานะปกติ (uid uid gid gid) เมื่อมีการเรียกใช้คำสั่งพิเศษซึ่งในที่นี้เป็นคำสั่งประเภท setuid โปรแกรม ทำให้สถานะของโปรเซสถูกเปลี่ยนเป็นสถานะพิเศษซึ่งก็คือ setuid state (uid **sid** gid gid) จากสถานะนี้ถ้าหากมีเหตุการณ์ที่ทำให้เกิดการเปลี่ยนแปลงของสถานะของโปรเซสไปเป็นสถานะ superuser state (**sid sid** gid gid) จะถือเป็นการบุกรุก เหตุการณ์ที่จะทำให้ค่า UID เปลี่ยนเป็น 0 จะต้องเกิดจากการเรียกใช้ system call *setreuid()* เท่านั้นซึ่งในการทำงานของ system call นี้จะทำการสลับค่าระหว่าง UID และ EUID นั่นคือกำหนดให้ค่าของ UID เป็น 0 ในขณะที่ค่าของ EUID เป็นค่าประจำตัวของผู้ใช้ทำให้ไม่เกิดกรณีที่ค่าของ UID และ EUID เป็น 0 ในเวลาเดียวกันหรือไม่ทำให้เกิดสถานะ superuser state ขึ้น เช่นเดียวกับในกรณีของการเปลี่ยนสถานะเป็น system group state ในภาพประกอบที่ 3.4 จาก setgid state (uid uid gid **sgid**) หากมีเหตุการณ์ที่ทำให้สถานะของโปรเซสเปลี่ยนจาก setgid state เป็น system group state (uid uid **sgid sgid**) ถือเป็นการบุกรุก ในกรณีที่ค่า GID เปลี่ยนเป็น 0 ยอมรับได้ในกรณีที่มีการเรียกใช้ system call *setregid()* เท่านั้น นั่นคือมีการสลับค่าระหว่าง GID และ EGID ซึ่งจะไม่ทำให้เกิดสถานะ system group state ขึ้นนั่นเอง



ภาพประกอบที่ 3.4 state transition diagram สำหรับการบุกรุกตามกฎข้อที่ 0 กรณีที่ 1



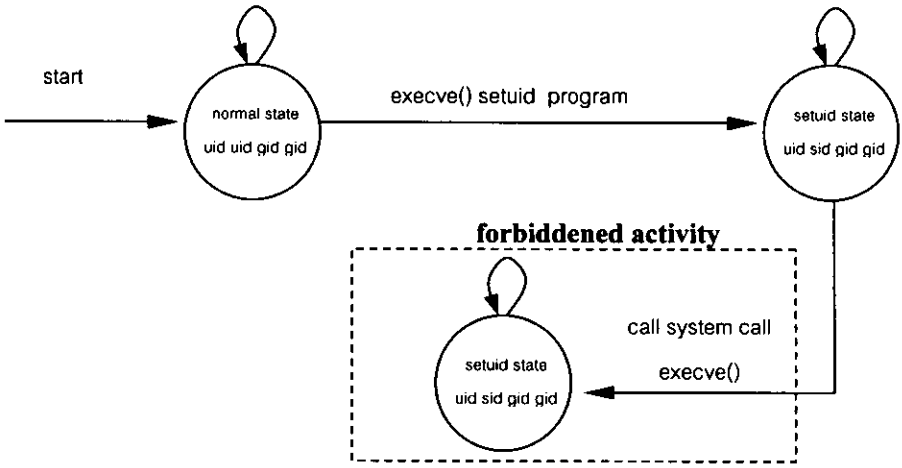
ภาพประกอบที่ 3.5 state transition diagram สำหรับการบุกรุกตามกฎข้อที่ 0 กรณีที่ 2

3.5.2 กฎข้อที่ 1 เมื่อโปรเซสอยู่ในสถานะสิทธิพิเศษ ไม่อนุญาตให้มีการเรียกใช้ system call *execve()*

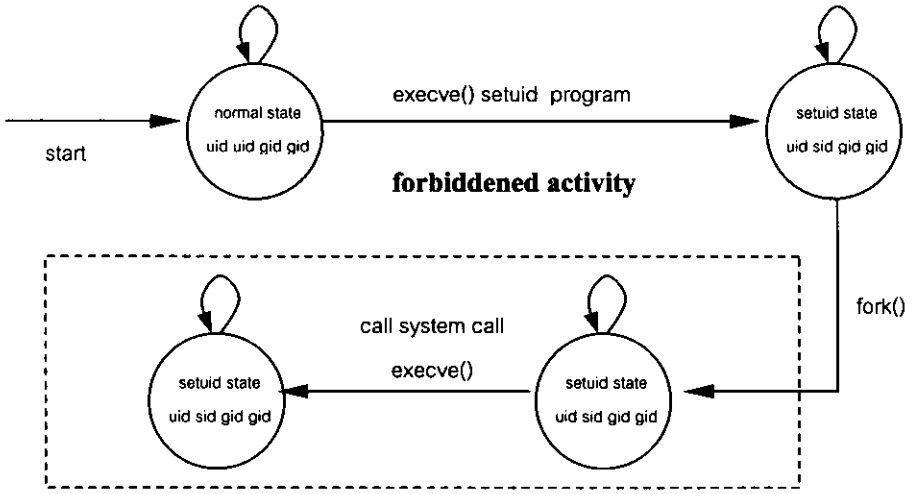
ผลกระทบที่เกิดขึ้น ถ้าหากผู้บุกรุกสามารถเรียกใช้ system call *execve()* ได้จะทำให้ผู้บุกรุกสามารถเรียกใช้โปรแกรมด้วยสิทธิของเจ้าของโปรแกรมซึ่งส่วนมากจะเป็นผู้ใช้ที่มีสิทธิสูงสุด เนื่องจากโปรแกรมที่เป็นข้อบกพร่องโดยส่วนใหญ่เป็นโปรแกรมประเภท setuid root เนื่องจากขณะที่ผู้บุกรุกเรียกใช้งานโปรแกรมนั้นตัวผู้บุกรุกมีสิทธิเทียบเท่าผู้ใช้สูงสุดนั่นคือโปรเซสขณะนั้น

มีค่า EUID เท่ากับ 0 การเรียกใช้โปรแกรมที่ทำให้เกิดปัญหา Buffer Overflow จะทำให้ผู้บุกรุกกลายเป็นผู้ใช้สูงสุดได้

การตรวจจับ จากภาพประกอบที่ 3.6 และ 3.7 เป็นการเรียกใช้ system call *execve()* ในขณะที่โปรเซสอยู่ในสถานะ setuid state โดยที่ในภาพประกอบ 3.6 เมื่อมีการเรียกใช้คำสั่งพิเศษประเภท setuid program สถานะของโปรเซสเปลี่ยนสถานะปกติเป็นสถานะที่มีสิทธิพิเศษ setuid state ต่อมามีการเรียกใช้ system call *execve()* ซึ่งสถานะในขณะนี้ยังคงอยู่ในสถานะพิเศษนั่นคือ ค่า EUID เป็น 0 ซึ่งมีสิทธิในการเรียกใช้ทรัพยากรที่ผู้ใช้สูงสุดมีสิทธิเรียกใช้ได้เนื่องจากในการเรียกใช้ทรัพยากรระบบจะพิจารณาจากค่า EUID เป็นหลัก ซึ่งจากจุดนี้ผู้บุกรุกสามารถเรียกใช้โปรแกรมของตัวเองซึ่งอาจจะก่อให้เกิดความเสียหายแก่ระบบได้จึงถือเป็นการบุกรุก ส่วนในกรณีของภาพประกอบที่ 3.7 มีการเรียกใช้ system call *fork()* ก่อนซึ่งทำให้โปรเซสที่เกิดขึ้นใหม่หรือโปรเซสลูกจะมีสถานะของโปรเซสตามสถานะของโปรเซสแม่นั่นก็คืออยู่ใน setuid state เช่นเดียวกัน เมื่อมีการเรียกใช้ system call *execve()* ก็จะเกิดผลกระทบเช่นเดียวกับกับในกรณีของภาพประกอบที่ 3.7



ภาพประกอบที่ 3.6 state transition diagram สำหรับการบุกรุกตามกฎข้อที่ 1 กรณีที่ 1



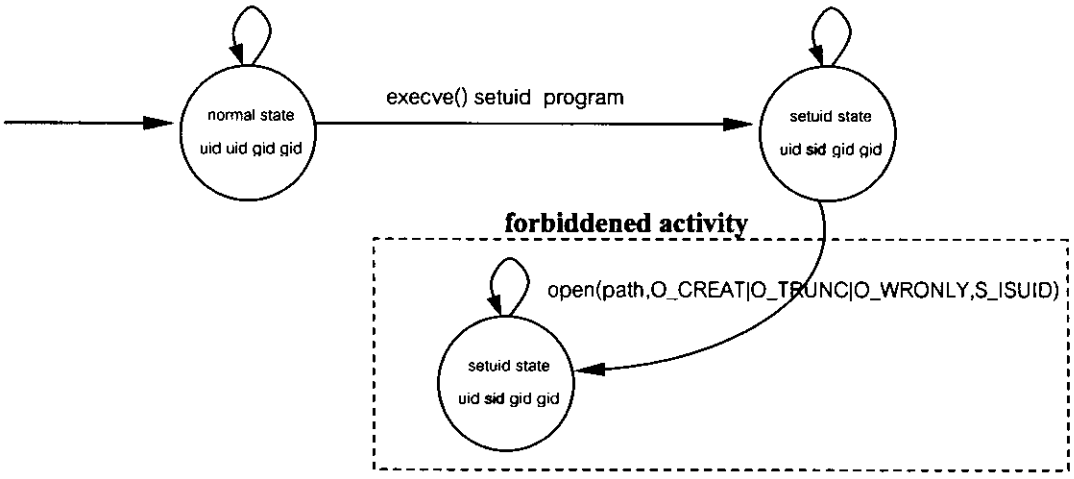
ภาพประกอบที่ 3.7 state transition diagram สำหรับการบุกรุกตามกฎข้อที่ 1 กรณีที่ 2

3.5.3 กฎข้อที่ 2 ขณะที่โปรเซสอยู่ในสถานะสิทธิพิเศษไม่อนุญาตให้มีการสร้าง setuid/setgid โปรแกรม อนุญาตเฉพาะ ผู้ใช้ที่มีสิทธิสูงสุดเท่านั้น

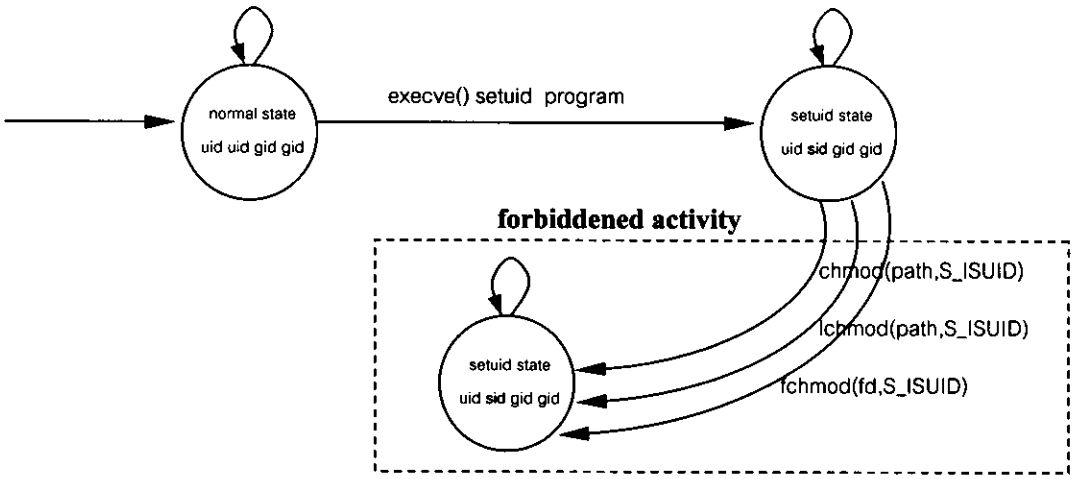
ผลกระทบที่เกิดขึ้น ถ้าผู้บุกรุกสามารถสร้าง setuid/setgid โปรแกรมได้ เนื่องจากในขณะที่ผู้ใช้อยู่ในสถานะสิทธิพิเศษโดยเฉพาะสถานะที่มีสิทธิเทียบเท่าผู้ใช้สูงสุดนี้ ผู้ใช้สามารถกำหนดให้โปรแกรมที่สร้างขึ้นเป็นโปรแกรมประเภท setuid root ได้และในที่สุดก็จะสามารถกระทำตนเป็นผู้ใช้สูงสุดได้จริง ๆ

การตรวจจับ การสร้างโปรแกรมประเภท setuid (set-user-id) หรือ setgid (set-group-id) โปรแกรมเป็นการสร้างแฟ้มที่มีการกำหนดสิทธิในการเรียกใช้แฟ้มนั้นซึ่งโดยส่วนมากแล้วจะเป็นการกำหนดให้มีสิทธิเทียบเท่าผู้ใช้สูงสุดในขณะที่เรียกใช้โปรแกรมนั้นยกตัวอย่างเช่นคำสั่ง *passwd* ซึ่งเป็นคำสั่งประเภท setuid โปรแกรมมีการกำหนดให้ในขณะที่เรียกใช้คำสั่งนี้จะต้องใช้สิทธิเทียบเท่าผู้ใช้สูงสุดเพื่อสามารถเรียกใช้ทรัพยากรที่จำเป็นได้ การสร้างโปรแกรมเหล่านี้สามารถกระทำได้โดย กรณีที่หนึ่งสร้างแฟ้มโดยเรียกใช้ system call *open(*const char **path*, int *oflag*, /* mode_t *mode* */... *)* และกำหนดโหมดเป็น S_ISUID หรือ S_ISGID ซึ่งเป็นการกำหนด UID หรือ GID เพื่อเรียกใช้โปรแกรม ในภาพประกอบที่ 3.8 เมื่อมีการเรียกใช้ setuid โปรแกรมจะเกิดการเปลี่ยนแปลงสถานะจาก normal state เป็น setuid state ซึ่งมีค่า EUID เท่ากับ 0 และมีการ

เรียกใช้ system call *open(*const char **path*, int *oflag*, /* mode_t *mode* */...) เพื่อสร้างแฟ้มโดยกำหนด mode เป็น S_ISUID ซึ่งทำให้ทุกครั้งที่มีการเรียกใช้แฟ้มที่ถูกสร้างขึ้นมาใหม่นี้โปรเซสในขณะที่เรียกใช้งานนั้นจะมีสิทธิเทียบเท่าผู้ใช้สูงสุดเหตุเนื่องจากขณะที่แฟ้มนี้ถูกสร้างขึ้นมาผู้ที่สร้างแฟ้มในขณะนั้นมีสถานะเทียบเท่าผู้ใช้สูงสุด (EUID=0) ซึ่งผู้ใช้สามารถที่จะกระทำการอันจะนำไปสู่การสร้างความเสียหายให้แก่ระบบได้ ดังเช่นกรณีที่มีการเรียกใช้แฟ้มซึ่งเป็นความลับของระบบเพื่อที่จะทำการลบหรือแก้ไขข้อมูลที่สำคัญ เพราะฉะนั้นการที่ผู้ใช้พยายามที่จะสร้างโปรแกรม setuid หรือ setgid ในขณะที่อยู่ในสถานะสิทธิพิเศษจึงถือเป็นการกระทำของผู้บุกรุกและถือเป็นกิจกรรมต้องห้าม สำหรับในกรณีที่สองการสร้าง setuid หรือ setgid โปรแกรม อาจจะกระทำได้โดยเรียกใช้ system call *chmod() lchmod()* หรือ *fchmod()* ซึ่ง system call เหล่านี้จะทำการกำหนดสิทธิการเข้าถึงหรือการใช้งานแฟ้มนั้นยกตัวอย่างเช่น ถ้ามีการกำหนดสิทธิการเข้าถึงของแฟ้มโดยกำหนดโหมดเป็น S_ISUID หรือ S_ISGID ก็จะเป็นการกำหนด UID หรือ GID ในการเรียกใช้แฟ้มนั้น



ภาพประกอบที่ 3.8 state transition diagram สำหรับการบุกรุกโดยสร้างโปรแกรมประเภท setuid กรณีที่ 1



ภาพประกอบที่ 3.9 state transition diagram สำหรับการบุกรุกโดยสร้างโปรแกรมประเภท setuid กรณีที่ 2

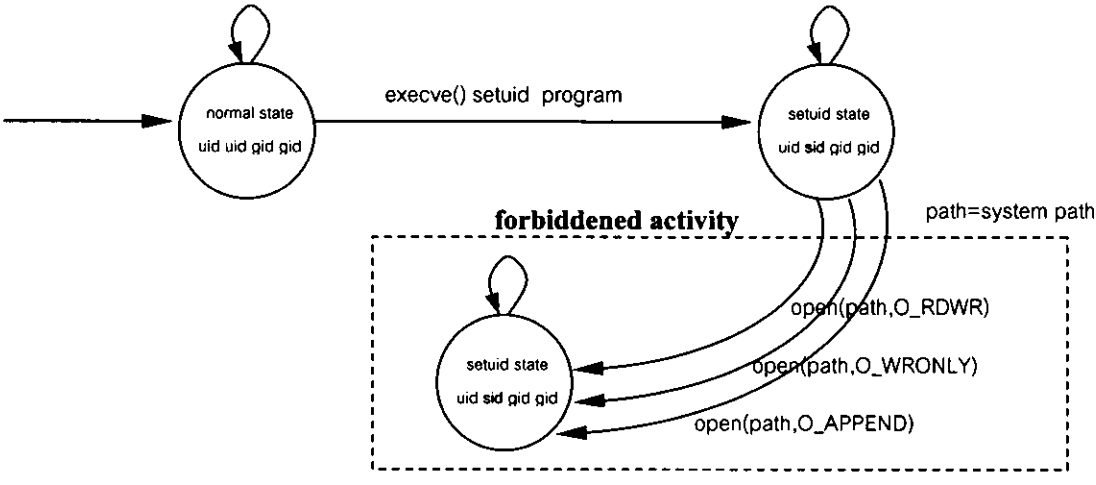
จากภาพประกอบที่ 3.9 จะเห็นได้ว่าเมื่อมีการเรียกใช้คำสั่งพิเศษซึ่งในที่นี้เป็นคำสั่งประเภท setuid ทำให้สถานะของโปรเซสเปลี่ยนจาก normal state เป็น setuid state และเมื่อมีการเรียกใช้ system call *chmod(), lchmod()* หรือ *fchmod()* โดยกำหนดโหมดเป็น S_SUID จะถือเป็นการบุกรุกเนื่องจากขณะนี้โปรเซสอยู่ใน setuid state ซึ่งค่า EUID ขณะนี้เป็น 0 นั่นคือถือว่าเจ้าของแฟ้มในขณะนี้มีสิทธิเทียบเท่า ผู้ใช้ที่มีสิทธิสูงสุด เพราะฉะนั้นเมื่อมีการกำหนดโหมดให้กับแฟ้มเป็น S_ISUID แฟ้มนั้นก็จะถูกกำหนดว่าเมื่อใดก็ตามที่มีการรันแฟ้มนี้ UID ในขณะที่รันแฟ้มนี้จะเป็น 0 นั่นคือถ้าในแฟ้มโปรแกรมนี้มีการกำหนดให้ทำการลบหรือแก้ไขแฟ้มใดก็ตาม ก็จะสามารถทำได้เนื่องจากมีสิทธิเทียบเท่า ผู้ใช้ที่มีสิทธิสูงสุด ซึ่งจะนำไปสู่การเกิดความเสียหายแก่ระบบได้

3.5.4 กฎข้อที่ 3 ไม่อนุญาตให้โปรเซสแก้ไขโปรแกรมระบบ

ผลกระทบที่เกิดขึ้น หากผู้บุกรุกสามารถแก้ไขโปรแกรมระบบได้ผู้บุกรุกจะสามารถฝังโค้ดประเภทม้าโทรจัน หรือเปิดช่องโหว่ให้กับโปรแกรมนั้นเพื่อใช้เป็นทางเข้าออกระบบได้

การตรวจจับ ผู้ใช้ทั่วไปโดยปกติแล้วไม่มีความจำเป็นจะต้องแก้ไขโปรแกรมระบบ ซึ่งโปรแกรมระบบในที่นี้หมายถึงโปรแกรมหรือคำสั่งที่ถูกติดตั้งมาพร้อมกับระบบหรือโปรแกรมที่ผู้ใช้สูงสุดเป็นผู้ติดตั้งอาทิเช่นโปรแกรมซึ่งเป็นต้นฉบับของคำสั่งที่ใช้ภายในระบบยกตัวอย่างเช่น

/usr/src/usr.bin/login/login.c เหล่านี้เป็นต้น ฉะนั้นเมื่อผู้ใช้อยู่ในสถานะที่มีสิทธิพิเศษอย่างเช่น setuid state ซึ่งมีสถานะเทียบเท่าผู้ใช้สูงสุดแล้วพยายามที่จะแก้ไขแฟ้มเหล่านี้จะถือเป็นการบุกรุก เนื่องจากผู้บุกรุกสามารถที่จะฝังโค้ด ในลักษณะของโทรจันไว้ในแฟ้มระบบเหล่านี้ได้ เราสามารถตรวจสอบได้จากโปรเซสในขณะนั้น โดยที่หากพบว่ามีการเรียกใช้แฟ้มซึ่งถูกกำหนดว่าเป็นโปรแกรมระบบ เพื่อที่จะทำการเขียนหรือแก้ไขนั่นคือมีการเรียกใช้ system call *open()* และมีการกำหนดค่าแฟลกเป็น O_RDWR O_WRONLY หรือ O_APPEND จะถือว่าเป็นการพยายามบุกรุกระบบและเป็นกิจกรรมต้องห้ามดังแสดงในภาพประกอบที่ 3.10 ซึ่งแสดงให้เห็นถึงกิจกรรมต้องห้ามเมื่อมีการพยายามแก้ไขแฟ้มโปรแกรมระบบ



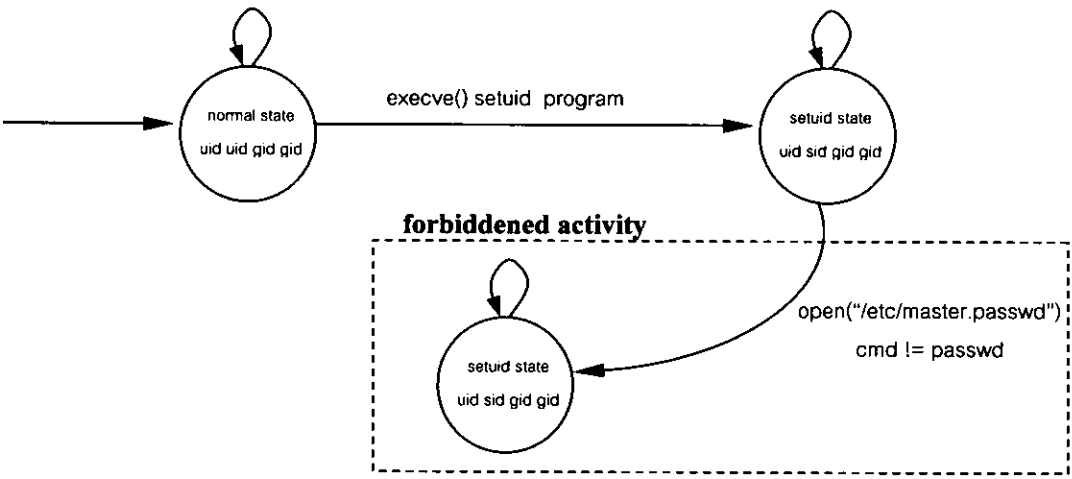
ภาพประกอบที่ 3.10 state transition diagram สำหรับการบุกรุกกรณีที่มีการแก้ไขแฟ้มโปรแกรมระบบ

3.5.5 กฎข้อที่ 4 ผู้ใช้ที่มีสิทธิสูงสุดเท่านั้นที่มีสิทธิในการสร้างบัญชีผู้ใช้ใหม่ได้

ผลกระทบที่เกิดขึ้น ถ้าหากผู้บุกรุกสามารถสร้างบัญชีรายชื่อผู้ใช้ใหม่ได้ผู้บุกรุกก็สามารถที่จะสร้างผู้ใช้ที่มีสิทธิสูงสุดซึ่งสามารถควบคุมการใช้งานทรัพยากรทั้งหมดในระบบได้

การตรวจจับ ในระบบปฏิบัติการยูนิกซ์ผู้ที่มีสิทธิในการสร้างบัญชีหรือเพิ่มผู้ใช้ใหม่เข้าสู่ระบบจะกระทำโดยผู้ใช้ที่มีสิทธิสูงสุดคำว่า "ผู้ใช้ที่มีสิทธิสูงสุด" ในที่นี้หมายถึงผู้ใช้ที่มีค่า

ของ UID EUID GID และ EGID ทุกค่าเท่ากับ 0 ตั้งแต่สถานะเริ่มต้นเพราะฉะนั้นในกรณีของผู้ใช้ปกติที่มีการเปลี่ยนสถานะของโปรเซสจากสถานะปกติ normal state เป็นสถานะพิเศษ setuid state ซึ่งมีสิทธิเทียบเท่าผู้ใช้ที่มีสิทธิสูงสุดแล้วพยามยามที่จะสร้างบัญชีผู้ใช้ใหม่จะถือว่าเป็นการบุกรุก การพิจารณากฎข้อนี้เนื่องจากในการสร้างบัญชีผู้ใช้ใหม่มีการเรียกใช้แฟ้มรหัสผ่านซึ่งเป็นแฟ้มชาโดว์เพื่อแก้ไข เช่นเดียวกันกับกระบวนการของการเปลี่ยนรหัสผ่านเพราะฉะนั้นเราจึงไม่สามารถใช้เงื่อนไขในการเรียกใช้แฟ้มเพียงอย่างเดียว สิ่งที่ใช้พิจารณาร่วมกันคือตัวคำสั่งในโปรเซสขณะที่มีการเรียกใช้แฟ้มนั้นหากเป็นคำสั่งที่ยกเว้น เนื่องจากระบบต้องใช้อย่างเช่นผู้ใช้ปกติเรียกใช้คำสั่ง passwd จะไม่ถือเป็นการบุกรุก จากภาพประกอบที่ 3.11 เมื่อโปรเซสเปลี่ยนสถานะจาก normal state เนื่องจากมีการเรียกใช้โปรแกรมประเภท setuid โปรแกรมไปเป็นสถานะ setuid state และมีการเรียกใช้ system call *open()* เพื่อเรียกใช้แฟ้มชาโดว์แต่เนื่องจากคำสั่งที่เรียกใช้แฟ้มในโปรเซสขณะนั้นเป็นคำสั่งอื่นซึ่งไม่ใช่คำสั่ง "passwd" ซึ่งผู้ใช้ปกติสามารถเรียกใช้ได้ จึงถือว่าเป็นการพยายามบุกรุก

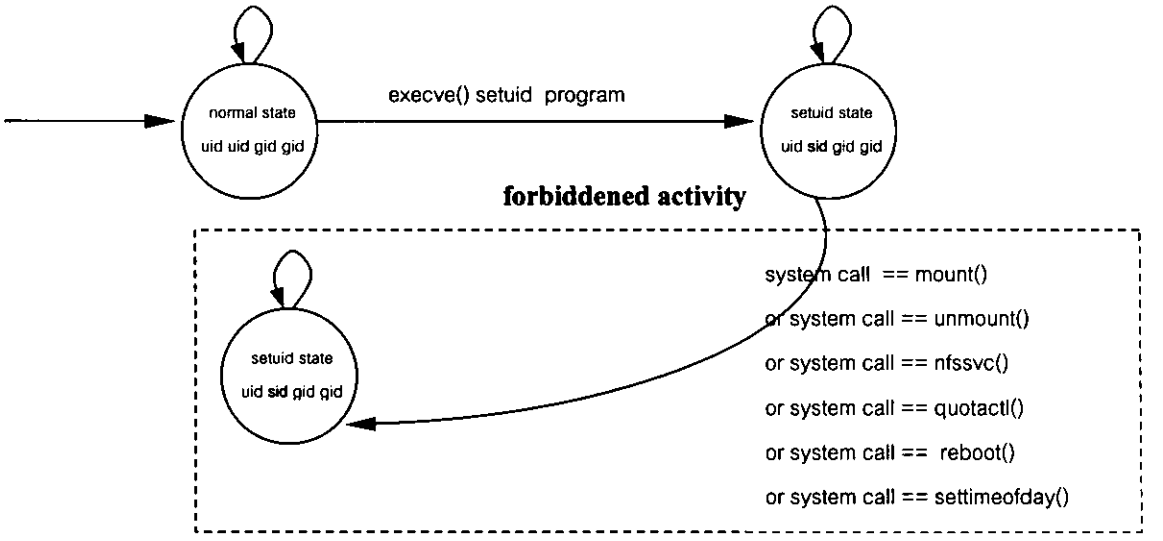


ภาพประกอบที่ 3.11 state transition diagram สำหรับการบุกรุกกรณีที่พยายามเรียกใช้แฟ้มรหัสผ่าน

3.5.6 กฎข้อที่ 5 เมื่อโปรเซสอยู่ในสถานะที่มีสิทธิพิเศษไม่อนุญาตให้มีการเรียกใช้ system call *mount(), unmount(), nfssvc(), quotactl(), reboot(), settimeofday(), swapon()* ซึ่งสามารถเรียกใช้ได้โดยผู้ใช้ที่มีสิทธิสูงสุดเท่านั้น

ผลกระทบที่เกิดขึ้น ถ้าหากผู้บุกรุกสามารถเรียกใช้งาน system call เหล่านี้ได้จะทำให้เกิดความเสียหายแก่ระบบได้ยกตัวอย่างเช่น การเรียกใช้ system call *mount()* ซึ่งเป็นการอนุญาตให้เรียกใช้แฟ้มระบบได้ ยกตัวอย่างเช่นมีการ mount procfs filesystem ซึ่งทำให้สามารถแก้ไขโปรเซสที่กำลังทำงานอยู่หรือเปลี่ยนแปลงระดับความปลอดภัยของระบบ

การตรวจจับ เนื่องจากเมื่อโปรเซสอยู่ในสถานะ setuid state หรือ setgid state จะมีสิทธิเทียบเท่าผู้ใช้ที่มีสิทธิพิเศษหรือกลุ่มระบบ ทำให้สามารถเรียกใช้ทรัพยากรในระบบได้เทียบเท่าผู้ใช้สูงสุด ดังนั้นจากกฎข้อนี้ system call บางตัวจึงถูกกำหนดให้ถูกเรียกใช้ได้จาก "ผู้ใช้ที่มีสิทธิสูงสุด" เท่านั้นเพื่อเลี่ยงความเสียหายที่จะเกิดขึ้นกับระบบหากพบว่า system call เหล่านี้ถูกเรียกใช้โดยโปรเซสที่อยู่ในสถานะสิทธิพิเศษนั่นคือ setuid state หรือ setgid state จะถือเป็นการบุกรุก เมื่อลองพิจารณา system call *mount()* ซึ่งเป็นการอนุญาตให้เรียกใช้แฟ้มระบบได้ ยกตัวอย่างเช่นมีการ mount procfs filesystem ซึ่งทำให้สามารถแก้ไขโปรเซสที่กำลังทำงานอยู่หรือเปลี่ยนแปลงระดับความปลอดภัยของระบบ โดยทั่วไปแล้วสามารถเข้าถึงได้เฉพาะเมื่อ EUID มีค่าเป็น 0 เท่านั้น หรือในกรณีของ system call *nfssvc()* ซึ่งเป็นฟังก์ชันที่ถูกเรียกใช้โดย NFS (Network File System) จะอนุญาตให้มีการส่งเข้าหรือส่งออกข้อมูลระหว่างโฮสต์ ดังนั้นโฮสต์อื่นสามารถเข้าถึงข้อมูลได้โดยผ่านเน็ตเวิร์กและทำให้ผู้บุกรุกอยู่ในสถานะที่สามารถถส่ง "RPC messages" ซึ่งก่อให้เกิด DoS (Denial of Service) โดยที่การโจมตีแบบ DoS โดยทั่วไปมีจุดประสงค์เพื่อป้องกันไม่ให้ใช้งานหรือทำให้เกิดความเสียหายต่อการใช้งานคอมพิวเตอร์หรือทรัพยากรของระบบเครือข่าย ภาพประกอบที่ 3.12 แสดงให้เห็นถึง state transition diagram สำหรับการบุกรุกในกรณีที่มีการเรียกใช้ system call ที่ไม่ได้รับอนุญาตในขณะที่โปรเซสอยู่ใน setuid state



ภาพประกอบที่ 3.12 state transition diagram สำหรับการบุรุกกรณีที่มีการเรียกใช้ system call ที่ไม่ได้รับการอนุญาต

3.5.7 สรุปผลการวิเคราะห์การบุกรุกและกฎที่ใช้สนับสนุนการตรวจจับ

จากการวิเคราะห์การบุกรุกที่กล่าวมา จะเห็นได้ว่าเหตุการณ์เกือบทั้งหมดสามารถเกิดขึ้นได้ในขณะที่โปรเซสอยู่ในสถานะที่มีสิทธิพิเศษ นั่นคือสถานะที่มีสิทธิเทียบเท่าผู้ใช้สูงสุด หรือสถานะที่มีสิทธิเทียบเท่ากลุ่มระบบ โดยเฉพาะสถานะที่มีสิทธิเทียบเท่าผู้ใช้สูงสุดหรือ setuid state ซึ่งสามารถใช้ทรัพยากรในระบบได้เทียบเท่ากับผู้ใช้ที่มีสิทธิสูงสุด กรณีการบุกรุกที่เกิดขึ้นสามารถนำไปสู่การก่อให้เกิดความเสียหายแก่ระบบทั้งในด้านการเกิดความเสียหายแก่ข้อมูล การกระทำในลักษณะของม้าโทรจัน นอกจากสถานะของโปรเซสแล้ว system call ที่ถูกเรียกใช้ในแต่ละโปรเซสรวมทั้งคำสั่งที่เรียกใช้ system call นั้น ๆ ยังเป็นเงื่อนไขที่ใช้พิจารณาร่วมด้วยเช่นกัน

ฉะนั้นจากข้อมูลเหล่านี้ถ้าหากเราสามารถสร้างโปรแกรมที่ใช้สำหรับตรวจจับการบุกรุกที่จะเกิดขึ้นตามลักษณะที่กล่าวมาในกฎทั้งหกข้อ ก็สามารถที่จะช่วยลดอัตราความเสียหายที่จะเกิดเป็นผลตามมาจากเหตุการณ์ที่เราตรวจจับได้ ในหัวข้อถัดไปจะกล่าวถึงการออกแบบโปรแกรมซึ่งครอบคลุมตามเงื่อนไขของกฎทั้งหกข้อโดยเริ่มจากการออกแบบข้อมูลเข้า การออกแบบโปรแกรมของกฎแต่ละข้อ และสุดท้ายการออกแบบโปรแกรมในภาพรวมของทั้งระบบ

3.6 การออกแบบข้อมูลนำเข้า

จากหัวข้อที่ 3.2 ซึ่งเป็นการวิเคราะห์ข้อมูลที่ได้จากการติดตามการเรียกใช้ system call ในคำสั่งปกติและคำสั่งพิเศษและหัวข้อ 3.5 ซึ่งกล่าวถึงกฎที่ใช้ในการพิจารณาการบุกรุก ข้อมูลเข้าที่เราใช้เพื่อนำมาพิจารณาเพื่อตรวจจับและตัดสินว่าเป็นการบุกรุกประกอบด้วย

1. หมายเลขโปรเซส
2. สถานะของโปรเซสซึ่งประกอบด้วยค่า uid euid gid และ egid
3. system call ที่ถูกเรียกใช้ พร้อมทั้งค่าพารามิเตอร์อาทิเช่น แฟ้มและตำแหน่งของแฟ้มที่ถูกเรียกใช้ แฟลก และโหมดของ system call ตัวนั้น
4. แฟ้มและตำแหน่งของแฟ้มที่ถูกเรียกใช้
5. คำสั่งที่เรียกใช้ system call

โดยที่ข้อมูลเหล่านี้ได้มาจากการเรียกใช้ system call *ktrace()* ซึ่งเป็นการทำงานในรูปแบบเดียวกับข้อมูลออกที่ได้จากตัวโปรแกรม ktrace แต่เลือกเอาเฉพาะส่วนที่สนใจนั่นคือมีการกำหนดค่า trpoints ดังนี้

```
trpoints |= KTRFAC_SYSTEM CALL | KTRFAC_SYSRET | KTRFAC_NAMEI | KTRFAC_INHERIT
```

ค่าที่กำหนดประกอบด้วย system call ที่ถูกเรียกใช้ (KTRFAC_SYSTEM CALL) ค่าที่ถูกส่งกลับมาโดย system call (KTRFAC_SYSRET) ชื่อแฟ้มและตำแหน่งแฟ้ม (KTRFAC_NAMEI) และสุดท้ายกำหนดให้มีการติดตามโปรเซสนั้นเมื่อโปรเซสมีการสร้างโปรเซสลูก (KTRFAC_INHERIT) ตามลำดับ หมายเลขโปรเซสที่ถูกติดตามได้มาจากโปรเซสที่กำลังทำงานอยู่ในขณะนั้นข้อมูลที่เข้ามาจะถูกอ่านตามลำดับบรรทัดตามตารางที่แสดงในหัวข้อที่ 3.1

ตัวอย่างข้อมูลนำเข้า

```
1000    0  100   100     10873   passwd  CALL    open(0x480708c0,0xa01,0x180)
1000    0  100   100     10873   passwd  NAMI    "/etc/ptmp"
1000    0  100   100     10873   passwd  RET     open 3
```

จากตัวอย่างข้อมูลด้านบนแต่ละบรรทัดคือข้อมูลเข้าที่ระบบตรวจจับการบุกรุกอ่านเข้ามาในแต่ละครั้ง โดยที่

1000 0 100 100	คือ	ค่าของ UID EUID GID EGID ของโปรเซส ตามลำดับ
10873	คือ	หมายเลขโปรเซส
passwd	คือ	คำสั่งที่ถูกติดตามการทำงาน
CALL	คือ	tracepoints ประเภท KTRFAC_SYSCALL
NAMI	คือ	tracepoints ประเภท KTRFAC_NAMI
open	คือ	system call ที่ถูกเรียกใช้
(0x480708c0,0xa01,0x180)	คือ	ค่า argument ของ system call *open()*
"/etc/ptmp"	คือ	ที่อยู่ของแฟ้มที่เกี่ยวข้องกับ system call ที่ถูกเรียกใช้
RET	คือ	tracepoints ประเภท KTRFAC_SYSRET
3	คือ	ค่าที่ถูกส่งกลับเมื่อเรียกใช้ system call

ในหัวข้อถัดไปจะกล่าวถึงการออกแบบโปรแกรมเพื่อตรวจจับการบุกรุกตามกฎที่ใช้สนับสนุนการบุกรุก ในแต่ละข้อ

3.7 การออกแบบการโปรแกรมที่ใช้ในการตรวจจับตามกฎแต่ละข้อ

การออกแบบโปรแกรมในเบื้องต้น เริ่มจากการออกแบบโปรแกรมเพื่อใช้ตรวจจับการบุกรุกตามกฎในแต่ละข้อเพื่อให้แน่ใจว่าสามารถใช้ตรวจจับได้จริงในกฎข้อนั้น ๆ โดยในขั้นแรกนี้จะยังไม่นำเรื่องของสถานะโปรเซสมาพิจารณาและทำการติดตามการทำงานของโปรเซสซึ่งเขียนจำลองสถานะการณ์ขึ้นมาตามกฎแต่ละข้ออธิบายได้ดังนี้

3.7.1 การออกแบบโปรแกรมการตรวจจับกฎข้อที่ 0

ในกระบวนการทำงานของ system call *setreuid()* หรือ *setregid()* จะกระทำการสลับค่าระหว่างค่า UID /GID กับค่าของ EUID/EGID ของโปรเซสในขณะนั้น ค่าของ UID/GID จะเปลี่ยนแปลงหลังจากที่ system call ที่ถูกเรียกใช้ส่งค่ากลับ (อาทิเช่น ขณะที่เรียกใช้ system call *setreuid*() โปรเซสมีสถานะเป็น setuid state (1000 0 100 100) หลังจากนั้นเมื่อมีการส่งค่ากลับจาก

system call *setreuid()* ค่าของ UID และ EUID จะเปลี่ยนเป็น (0 1000 100 100) เพราะฉะนั้นในการออกแบบโปรแกรมในส่วนนี้ในขณะที่โปรแกรมอ่านข้อมูลเข้ามาแต่ละครั้ง เมื่อมีการเรียกใช้ system call โปรแกรมจะทำการเก็บค่า system call ที่ถูกเรียกใช้ไว้ในตัวแปร sys_curr ขั้นต่อไปเมื่อมีการส่งค่ากลับ ค่าของ UID/GID จะถูกพิจารณาถ้าหากพบว่าค่า UID หรือ GID เปลี่ยนเป็นศูนย์ ค่าที่ถูกเก็บไว้ในตัวแปร sys_curr จะนำมาพิจารณาร่วมด้วย นั่นคือ ถ้าค่าของ sys_curr ไม่ใช่ SYS_setreuid หรือ SYS_setregid จะถือว่าเป็นการบุกรุก

3.7.2 การออกแบบโปรแกรมการตรวจจับกฎข้อที่ 1

จากการวิเคราะห์การตรวจจับกฎข้อที่ 1 ในหัวข้อ 3.5.2 สิ่งที่นำมาพิจารณาคือการเรียกใช้ system call *execve()* ฉะนั้นการออกแบบโปรแกรมสามารถกระทำได้โดยเมื่อมีข้อมูลเข้าซึ่งเป็นส่วนของ KTRFAC_SYSCALL ตัวแปร sys_curr จะถูกกำหนดให้รับค่านั้น หลังจากนั้นค่า sys_curr จะถูกตรวจสอบว่าเป็น system call *execve()* หรือไม่ ถ้าพบว่าใช่ จะถือว่าเป็นการบุกรุก

3.7.3 การออกแบบโปรแกรมการตรวจจับกฎข้อที่ 2

สำหรับการออกแบบในกฎข้อที่ 2 ซึ่งเป็นการตรวจจับการสร้างแฟ้มประเภท setuid/setgid โปรแกรม สามารถแยกพิจารณาได้เป็นสองกรณีคือ ในกรณีที่หนึ่งเป็นการตรวจสอบการสร้างแฟ้มใหม่โดยการกำหนดโหมด และในกรณีที่สองคือการเปลี่ยนโหมดแฟ้มที่มีอยู่แล้ว สำหรับในส่วนที่หนึ่งนั้น system call ที่ถูกพิจารณาคือ system call *open()* เนื่องจากในการสร้างแฟ้มนั้นเป็นการเรียก system call *open()* โดยมีการกำหนดแฟลกเป็น O_CREAT | O_TRUNC | O_WRONLY ขั้นตอนในการตรวจจับจะเริ่มจากเมื่อ system call *open()* ถูกเรียกใช้ซึ่งตรวจสอบได้จากข้อมูลที่ติดตามการทำงานในส่วนของ KTRFAC_SYSTEM CALL หลังจากที่ตรวจจับได้ว่า system call ที่ถูกเรียกใช้คือ system call *open()* ขั้นตอนถัดมาคือการพิจารณาแฟลก ของ system call *open()* หาก แฟลกที่ถูกกำหนดเป็นแฟลกที่กำหนดเพื่อใช้ในการสร้างแฟ้ม สิ่งที่จะต้องพิจารณาร่วมซึ่งจะเป็นสิ่งที่ชี้ให้เห็นว่าเป็นการสร้างแฟ้มประเภท setuid/setgid โปรแกรมก็คือโหมดที่กำหนดใน system call *open()* โดยที่โหมดที่เราสนใจและถือว่าการกระทำนี้เข้าข่ายการบุกรุกคือโหมด S_ISUID และ โหมด S_ISGID ซึ่งเป็นโหมดที่ใช้กำหนดผู้ใช้และกลุ่มผู้ใช้ หลังจากที่ตรวจจับได้โปรแกรมจะทำการเรียก system call *exit()* เพื่อสิ้นสุดการทำงานของโปรแกรมการตรวจจับ สำหรับในกรณีที่สอง system call ที่ถูกพิจารณาคือ system call *chmod() lchmod()*

และ *fchmod()* ซึ่ง system call ทั้งสามนี้ทำงานเหมือนกันคือเปลี่ยนโหมดแฟ้มต่างกันตรงพารามิเตอร์ที่กำหนดแฟ้มจะเป็นรูปแบบของการระบุตำแหน่งแฟ้มและชื่อแฟ้ม การระบุแฟ้มแบบซอฟต์ลิงค์ และการกำหนดแฟ้มโดยใช้ค่า fd ซึ่งถ้าหากโหมดในการเรียกใช้ system call เหล่านี้ถูกกำหนดเป็น S_ISUID หรือ S_ISGID ก็จะถือว่าเป็นการกระทำที่เข้าข่ายการบุกรุกโปรแกรมจะเรียกใช้ system call *exit()* เพื่อออกจากโปรแกรมการตรวจจับเช่นเดียวกัน

3.7.4 การออกแบบโปรแกรมการตรวจจับกฎข้อที่ 3

จากการวิเคราะห์กฎที่ใช้สนับสนุนการตรวจจับข้อที่ 3 ในหัวข้อ 3.5.4 จะเห็นได้ว่าสิ่งที่ใช้ประกอบการพิจารณานอกจากเรื่องของ system call แล้วส่วนประกอบอีกประการหนึ่งคือเรื่องของแฟ้มระบบ ฉะนั้นในการออกแบบการตรวจจับกฎข้อนี้จะต้องมีการนิยามแฟ้มที่ถือว่าเป็น *แฟ้มระบบ* ในเบื้องต้นของการออกแบบโปรแกรมจะกำหนดแฟ้มที่ถือว่าเป็น*แฟ้มระบบ* ในตัวโปรแกรมเพราะจุดประสงค์หลักของการออกแบบโปรแกรมการรวจจับในแต่ละข้อเพื่อเน้นว่าโปรแกรมสามารถตรวจจับได้จริง การตรวจจับจะเริ่มจากการพิจารณาข้อมูลเมื่อมีการเรียกใช้ system call *open()* แฟ้มที่ถูกเรียกใช้จะถูกพิจารณาเป็นอันดับแรก ถ้าหากพบว่าแฟ้มที่ถูกเรียกใช้เป็นแฟ้มระบบค่าแฟลกที่ถูกกำหนดใน system call *open()* จะถูกพิจารณานั่นคืออนุญาตให้กำหนดแฟลกเป็น O_RDONLY หรือ อนุญาตให้เปิดแฟ้มแบบอ่านอย่างเดียวเท่านั้น ถ้าหากตรวจจับได้ว่าแฟลกที่กำหนดผิดไปจากนี้ นั่นคือมีการกำหนดแฟลกเป็น O_WRONLY O_RDWR O_APPEND โปรแกรมจะเรียก system call *exit()* เพื่อหยุดการทำงานการตรวจจับ

3.7.5 การออกแบบโปรแกรมการตรวจจับกฎข้อที่ 4

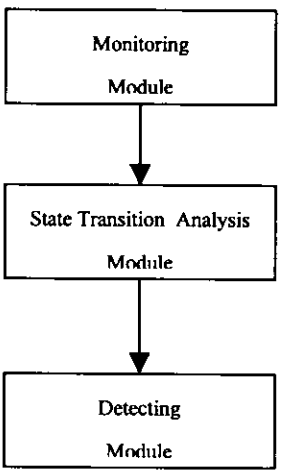
สำหรับในการออกแบบโปรแกรมสำหรับกฎข้อที่สี่นี้เป็นการตรวจสอบคำสั่งที่เรียกใช้แฟ้มชาโดว์โดยกำหนดให้เฉพาะคำสั่ง *passwd* เท่านั้นที่สามารถเรียกใช้งานแฟ้มนี้ได้ ฉะนั้นในการออกแบบโปรแกรมสิ่งที่ต้องตรวจสอบคือ ขั้นแรกเมื่อมีการเรียกใช้ system call *open()* และตรวจสอบพบว่าแฟ้มที่ถูกเรียกใช้คือแฟ้มชาโดว์ (/etc/shadow) คำสั่งที่เรียกใช้แฟ้มจะถูกตรวจจับหากพบว่าไม่ใช่คำสั่ง *passwd* จะถือว่าเข้าข่ายการบุกรุก โปรแกรมจะเรียก system call *exit()* เพื่อหยุดการทำงานการตรวจจับ

3.7.6 การออกแบบโปรแกรมการตรวจจับกฎข้อที่ 5

การออกแบบโปรแกรมเพื่อตรวจจับการบุกรุกในกรณีการเรียกใช้ system call ที่ถูกกำหนดไว้ให้เรียกใช้ได้เฉพาะ *ผู้ใช้ที่มีสิทธิสูงสุด* นั้นสามารถตรวจสอบได้ในขั้นตอนแรกที่มีการรับข้อมูลเข้าและประเภทของการติดตามเป็นส่วนของ KTRFAC_SYSTEM CALL ซึ่งสามารถใช้ switch case ในการพิจารณา system call เหล่านี้ หากพบว่ามีการเรียกใช้ system call เหล่านี้ไม่ว่าจะเป็นตัวใด โปรแกรมจะเรียก system call *exit()* เพื่อออกจากโปรแกรมเช่นเดียวกับกฎข้ออื่น และเนื่องจากในระบบปฏิบัติการที่ใช้ในการทำงานวิจัยชิ้นนี้ system call ที่ใช้สำหรับการเพิ่ม/ลบ swap devices คือ *swapctl()* ไม่ใช่ตัว *swapon()* จึงใช้การตรวจจับ *swapctl()* แทน

3.8 การออกแบบการโปรแกรมที่ใช้ในการตรวจจับทั้งระบบ

จากการออกแบบการตรวจจับตามกฎแต่ละข้อ เมื่อนำมาใช้ในการออกแบบการตรวจจับทั้งระบบ การทำงานของกระบวนการตรวจจับโปรแกรมแบ่งออกเป็น 3 ส่วนดังภาพประกอบที่ 3.13 โดยที่ในส่วนของ Monitor โมดูล ทำหน้าที่ในการติดตามการทำงานของ system call โดยรับข้อมูลที่เกี่ยวข้องกับโปรเซสซึ่งได้แก่ User Credentials, System call, คำสั่ง, หมายเลขโปรเซส หลังจากนั้นข้อมูลเหล่านี้จะถูกส่งไปยัง State Transition Analysis โมดูล ในส่วนนี้ จะตรวจสอบสถานะของโปรเซสถ้าหากพบว่าโปรเซสอยู่ในสถานะที่มีสิทธิพิเศษข้อมูลจะถูกส่งไปยังโมดูลสุดท้ายคือ Detection โมดูลทำหน้าที่วิเคราะห์ว่าเหตุการณ์ที่เกิดขึ้นขณะนั้นเข้าข่ายการบุกรุกหรือไม่ และหากเข้าข่ายการบุกรุกจะทำการบันทึกข้อมูลไว้ในส่วนของแฟ้มบันทึกข้อมูลการบุกรุก



ภาพประกอบที่ 3.13 System Module

3.9 แผนภาพกระแสข้อมูล (Data Flow Diagram: DFD)

ในหัวข้อนี้จะแสดงแผนภาพกระแสข้อมูลหรือ DFD ของระบบตรวจจับการบุกรุกที่ออกแบบขึ้น โดยจะเริ่มจาก DFD ระดับสูงสุด (Context Level Data Flow Diagram) ภาพรวม DFD (Overview DFD) ของการตรวจจับการบุกรุก และ DFD ระดับย่อย (Lower-Level Data Flow Diagrams) ซึ่งแสดงแผนภาพกระแสข้อมูลในกระบวนการย่อยของระบบ

3.9.1 DFD ระดับสูง (Context Level Data Flow Diagram)

พจนานุกรมข้อมูลที่ใช้สำหรับอธิบาย Context Level Data Flow Diagram ซึ่งแสดงในภาพประกอบที่ 3.14 ถึงภาพประกอบที่ 3.21 มีความหมายดังต่อไปนี้

ข้อมูลโปรเซสที่เข้าข่ายการบุกรุก หมายถึง

หมายเลขโปรเซสที่ถูกติดตาม + กฎที่ถูกละเมิด + หมายเลข system call + หมายเลขเซสชัน + หมายเลขโปรเซส

ค่าที่เกี่ยวข้องกับ user หมายถึง

ค่า UID EUID GID EGID ของโปรเซสขณะนั้น

ค่าที่เกี่ยวข้องกับ system call หมายถึง

หมายเลขโปรเซส + trpoint + command + path_name + flag + mode

สถานะพิเศษ หมายถึง

EUID หรือ EGID มีค่าเท่ากับ 0

system call เข้าข่าย กฎข้อที่ 5 หมายถึง

system call *mount(), umount(), nfssvc(), quotactl(), reboot(), settimeofday(), swapon()*

command ที่ยกเว้น หมายถึง

คำสั่งที่กำหนดไว้ในแฟ้ม chkr2r3.dat ซึ่งเป็นคำสั่งที่ยกเว้นการตรวจสอบตาม กฎข้อที่ 2 และกฎข้อที่ 3

system program ที่กำหนด หมายถึง

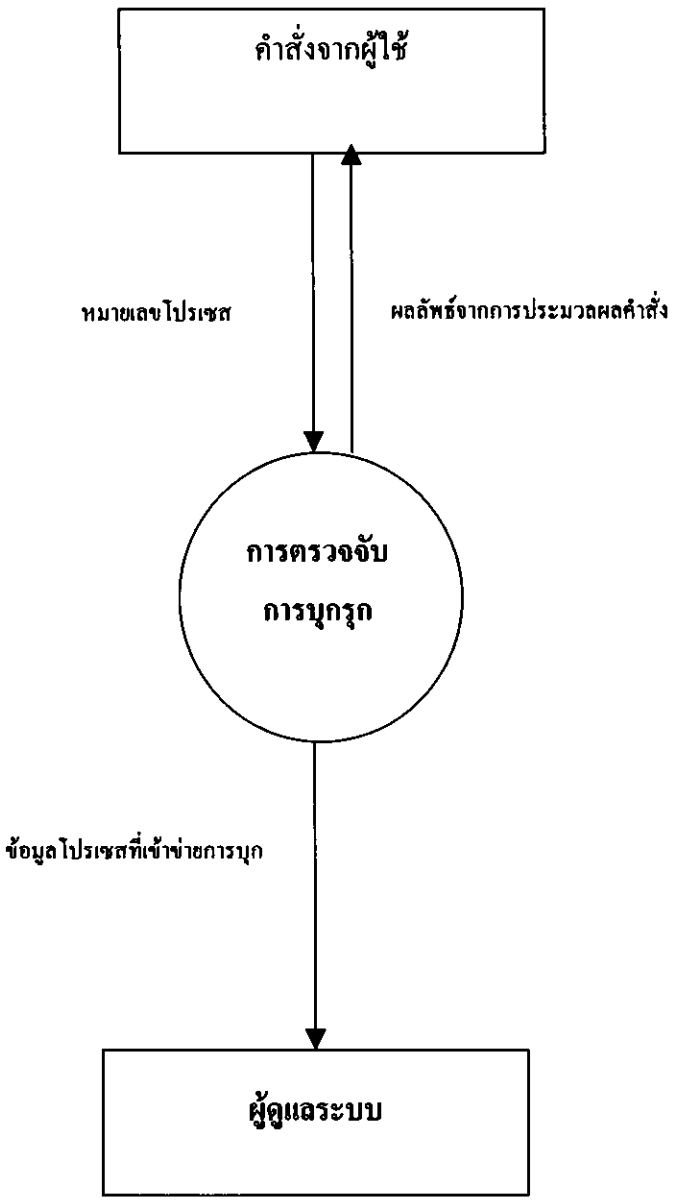
ข้อมูลที่อ้างถึงโปรแกรมระบบซึ่งอยู่ในไฟล์ chkr2r3.dat

แฟ้มบันทึกการบุกรุก หมายถึง

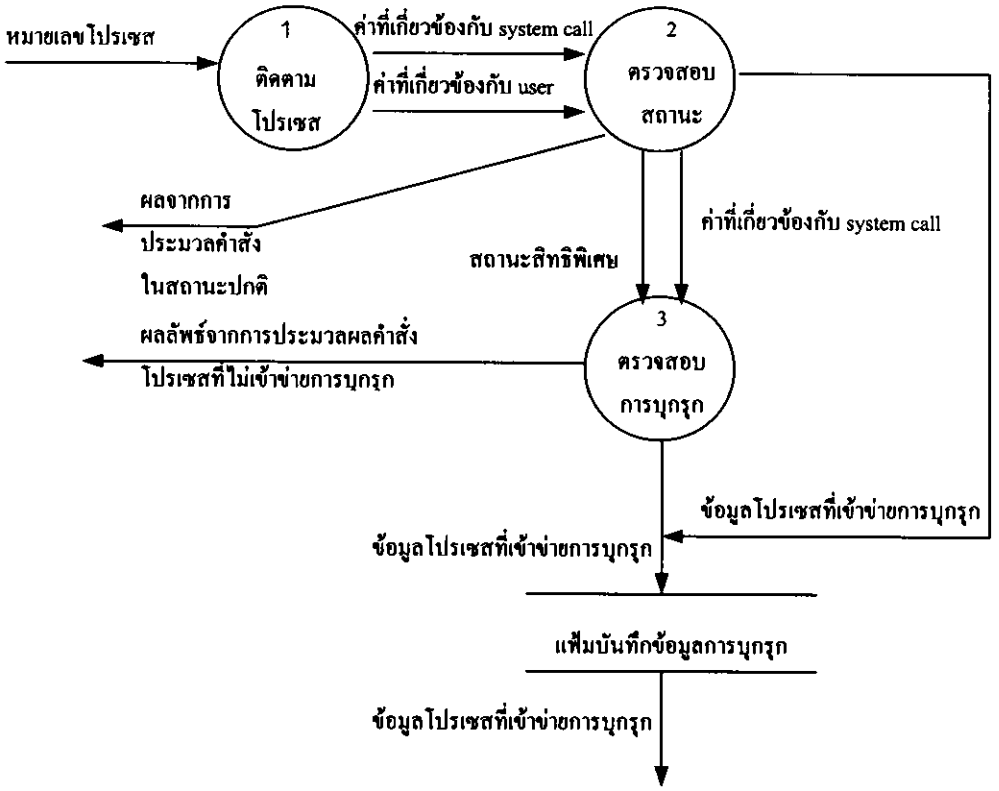
แฟ้ม /var/log/ids.log

ผลลัพธ์จากการประมวลผลคำสั่ง หมายถึง

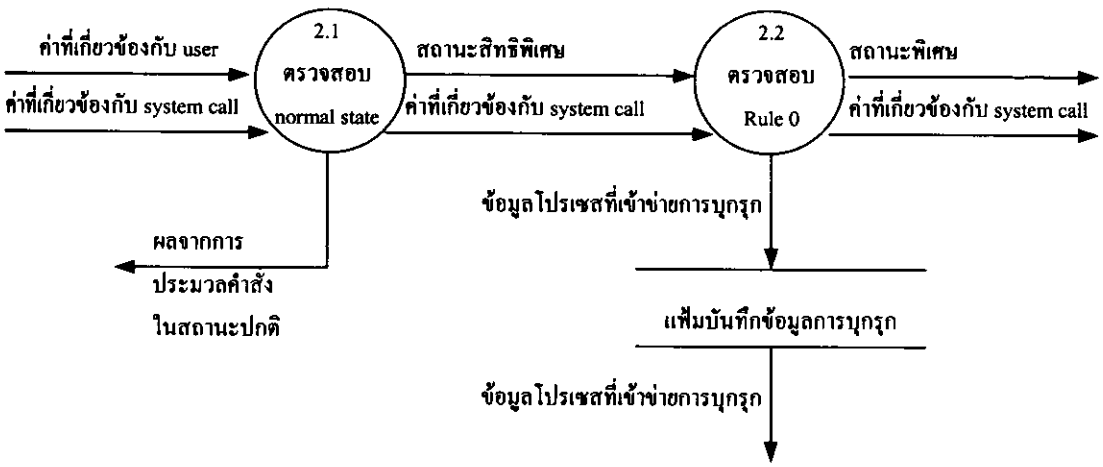
ผลลัพธ์จากการประมวลผลคำสั่งเมื่อไม่มีโปรเซสที่เข้าข่ายการบุกรุกในกระบวนการทำงานของคำสั่งนั้น



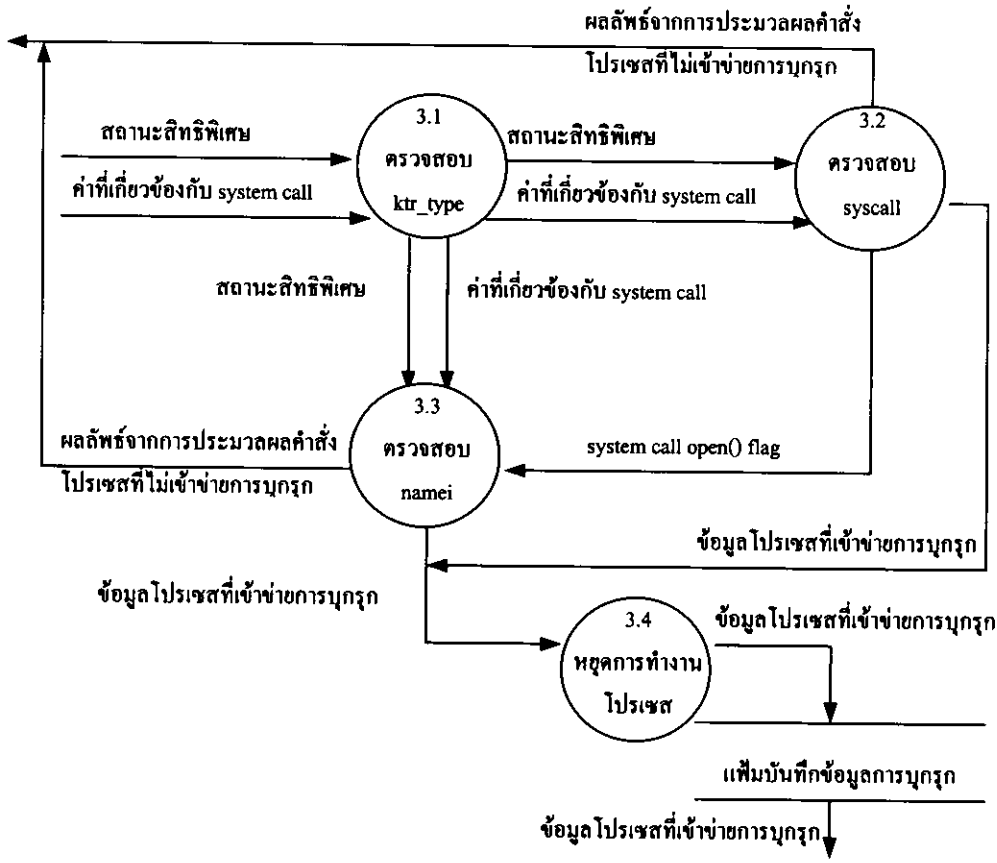
ภาพประกอบที่ 3.14 DFD ระดับสูงระบบตรวจจับการบุกรุก



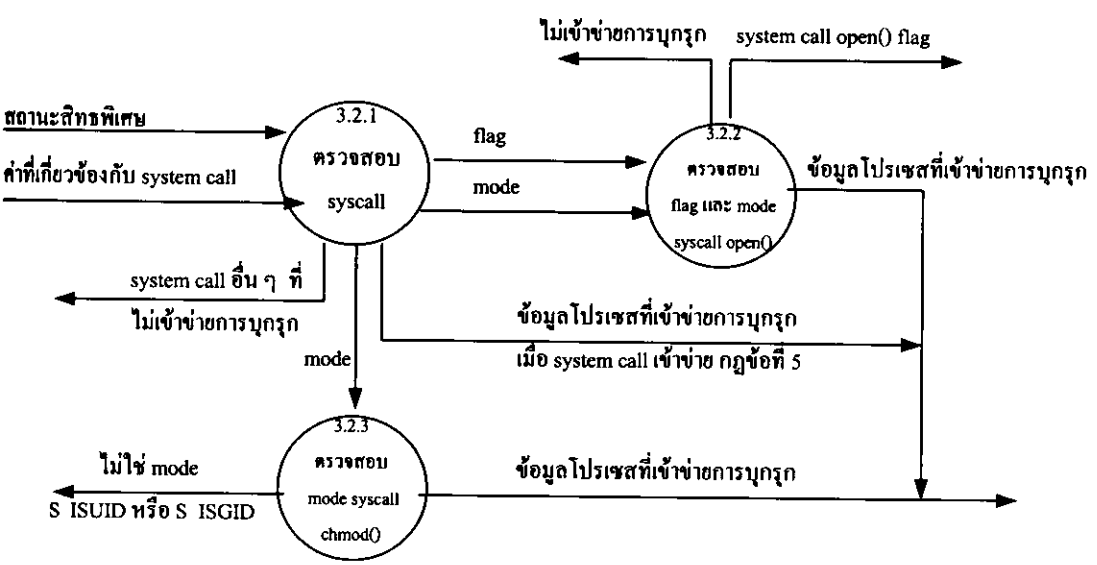
ภาพประกอบที่ 3.15 ภาพรวม DFD ระบบตรวจจับการบุกรุก



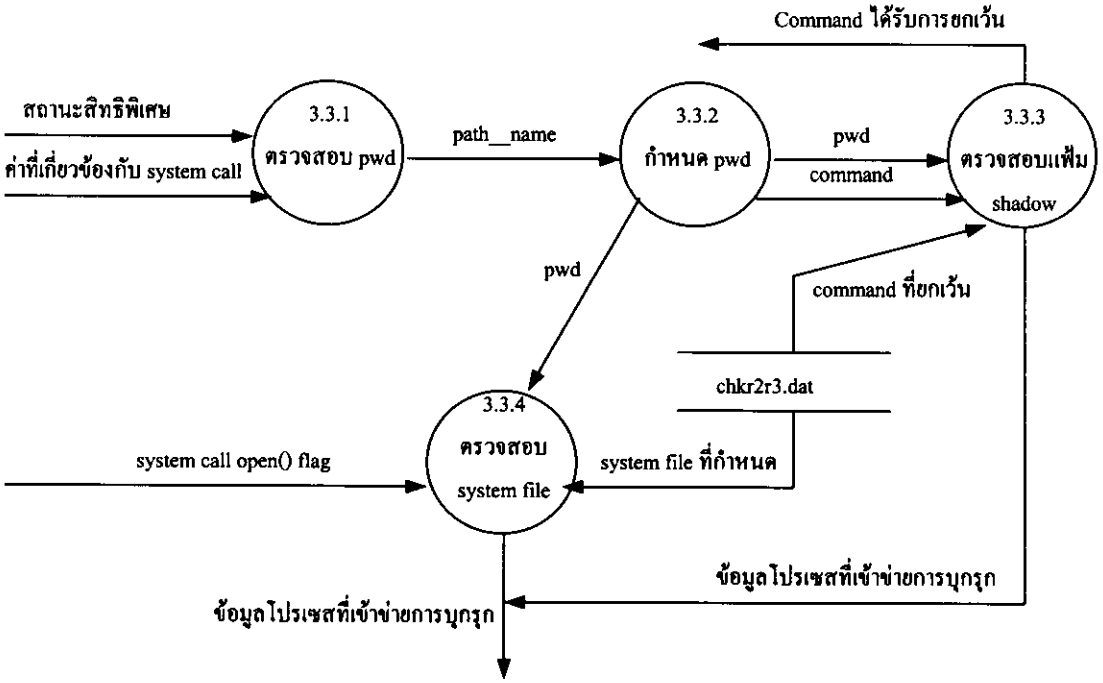
ภาพประกอบที่ 3.16 DFD ระดับย่อยของโปรเซสที่ 2 การตรวจสอบสถานะ



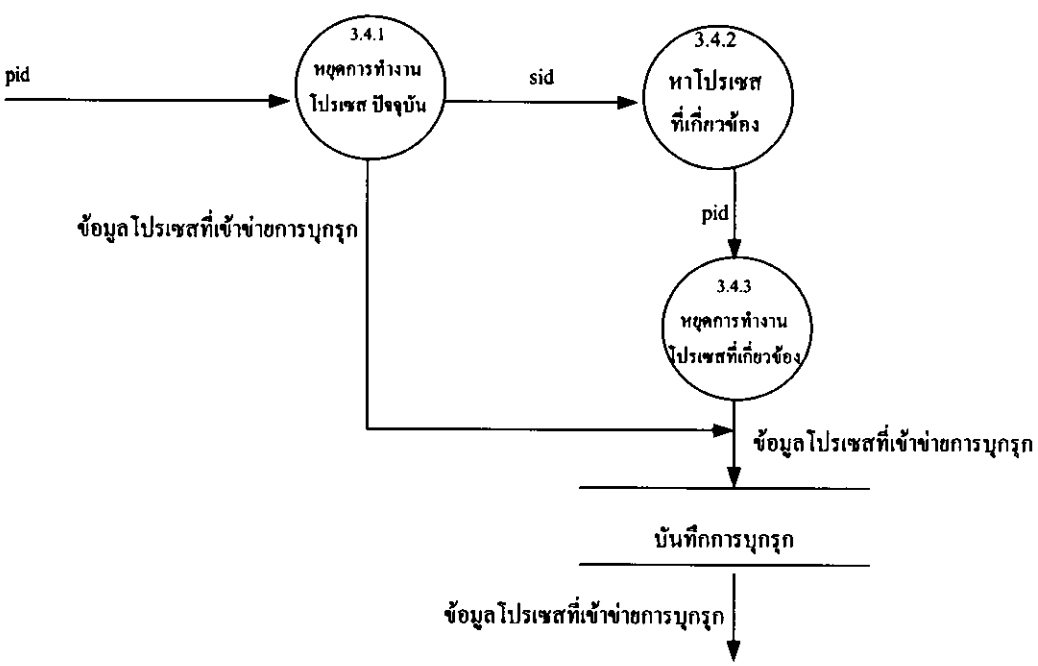
ภาพประกอบที่ 3.17 DFD ระดับย่อยของโปรเซสที่ 3 การตรวจจับการบุกรุก



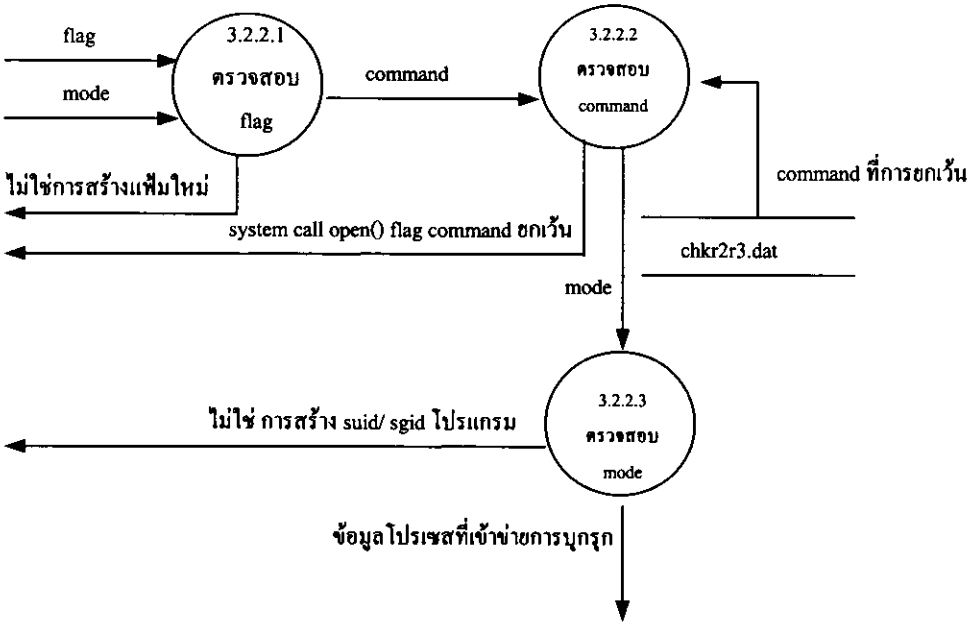
ภาพประกอบที่ 3.18 DFD ระดับย่อยของโปรเซสที่ 3.2 การตรวจสอบ trpoint ประเภท system call



ภาพประกอบที่ 3.19 DFD ระดับย่อยของโปรเซสที่ 3.3 การตรวจสอบ trpoint ประเภท krtr_namei



ภาพประกอบที่ 3.20 DFD ระดับย่อยของโปรเซสที่ 3.4 การหยุดการทำงานของโปรเซสที่เข้าข่ายการบุกรุก



ภาพประกอบที่ 3.21 DFD ระดับย่อยของโปรเซสที่ 3.2.2 การตรวจสอบ flag และ mode ของ system call *open()*

3.10 กลไกสถานะ (Finite State Machine: FSM)

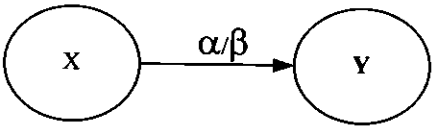
สำหรับในหัวข้อนี้จะแสดงให้เห็นการทำงานของระบบโดยใช้กลไกสถานะอธิบายการตรวจจับการบุกรุก ดังรายละเอียดดังนี้

เมื่อมีการเปลี่ยนแปลงสถานะจากสถานะหนึ่งไปยังอีกสถานะหนึ่งจะใช้สัญลักษณ์ α/β โดยที่

α จะใช้แทนเหตุการณ์ที่ทำให้เกิดการเปลี่ยนแปลง

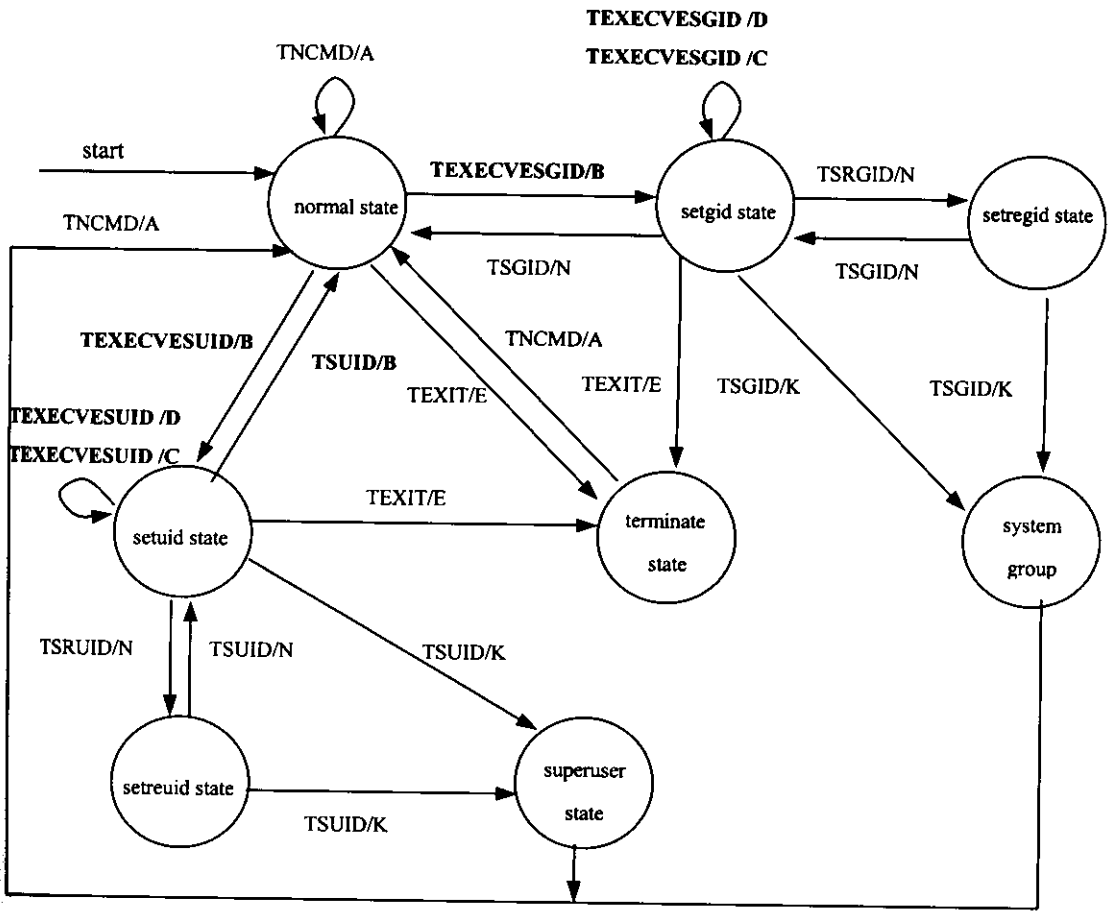
และ β จะใช้แทนกระบวนการทำงานที่จะกระทำเมื่อเกิดการเปลี่ยนแปลง

จากภาพประกอบที่ 3.22 สัญลักษณ์ α/β บนการเปลี่ยนแปลงจากสถานะ X ไปยังสถานะ Y หมายความว่าถ้าเหตุการณ์ α มาถึงในขณะที่อยู่ในสถานะ X ให้ไปทำกระบวนการ β และเปลี่ยนสถานะไปยังสถานะ Y



ภาพประกอบที่ 3.22 กลไกสถานะ

3.10.1 กลไกสถานะการทำงานของระบบเมื่อมีการเรียกใช้ system call



ภาพประกอบที่ 3.23 กลไกสถานะของกระบวนการตรวจจับการบุกรุกเมื่อมีเหตุการณ์ทำให้เกิดการเปลี่ยนแปลงของค่า UID EUID GID หรือ EGID ของโปรเซส

จากภาพประกอบประกอบที่ 3.23 ระบบตรวจจับการบุกรุกจะเริ่มทำงานในขณะที่โปรเซส อยู่ในสถานะ normal state หากไม่มีเหตุการณ์ที่ทำให้เกิดการเปลี่ยนแปลงสถานะ ระบบจะยังทำกระบวนการ A ซึ่งเป็นการอ่านข้อมลโปรเซสเข้ามาเรื่อย ๆ แต่เมื่อมีเหตุการณ์ TEXECVESUID เกิดขึ้นระบบจะทำกระบวนการ B และเปลี่ยนไปยังสถานะ setgid state และในสถานะ นี้จะมีการกระทำกระบวนการ C และกระบวนการ D หากในขณะที่อยู่ในสถานะ setuid นี้มีเหตุการณ์ TSUID เกิดขึ้นทำให้สถานะเปลี่ยนเป็น superuser state จะมีการทำกระบวนการ K คือหยุดการทำงานของโปรเซสนั้น แล้วกระบวนการนั้นจะกลับสู่ normal state เมื่อมีการเรียกใช้คำสั่งหรือเมื่อมีโปรเซสเกิดขึ้น นั่นคือมีเหตุการณ์ TNCMD ระบบจะทำกระบวนการ A เช่นเดียวกันกับกรณีที่มีเหตุการณ์ TEXIT นั่นคือจบกระบวนการของคำสั่งที่เริ่มมาจาก normal state ระบบจะกระทำกระบวนการ A เมื่อมีเหตุการณ์ TNCMD นั่นคือมีการเริ่มต้นคำสั่งใหม่โดยที่สถานะยังคงเป็น normal state กลไกสถานะสำหรับระบบตรวจจับนี้จะประกอบด้วยสถานะทั้งหมด 8 สถานะ คือ normal state, suid state, setreuid state, setgid state, setregid state, superuser state, system group state, terminate state และมีการบวนการทำงานหลัก ๆ คือ A, B, C, D, และ K ดังแสดงในภาพประกอบที่ 3.23 เหตุการณ์ที่ทำให้เกิดการเปลี่ยนแปลงได้แก่

TNCMD = การเรียกใช้คำสั่งปกติ

TEXECVSUID = การเรียกใช้โปรแกรมประเภท suid

TEXECVESGID = การเรียกใช้โปรแกรมประเภท sgid

TSUID = เรียกใช้ system call *setuid()*

TSGID = เรียกใช้ system call *setgid()*

TSRUID = เรียกใช้ system call *setreuid()*

TSRGID = เรียกใช้ system call *setreigd()*

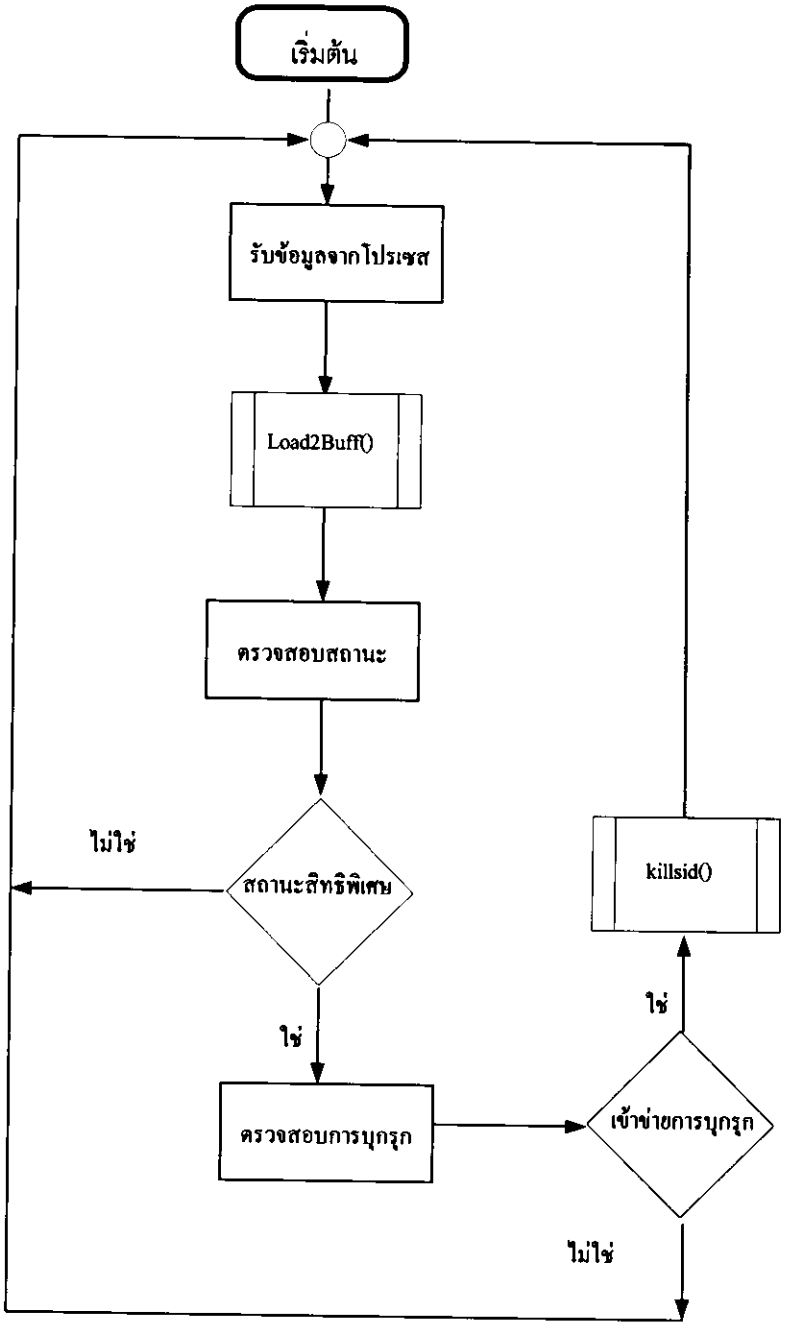
TEXIT = เรียกใช้ system call *exit()*

โดยมีกระบวนการทำงานคือ

A = fread_tail()	D=ktrnamei()
B=chk_rule_0	N=do_nothing
C=ktrsyscall()	K=killdetect()

หัวข้อถัดไปจะแสดงผังระบบงาน (System Flow Chart) ซึ่งแสดงการทำงานของโปรแกรมต่าง ๆ โดยจะเริ่มจาก ภาพประกอบที่ 3.24 ซึ่งแสดงผังระบบงานรวม ต่อด้วยผังงานของแต่ละโปรแกรมที่ถูกเรียกใช้ดังนี้

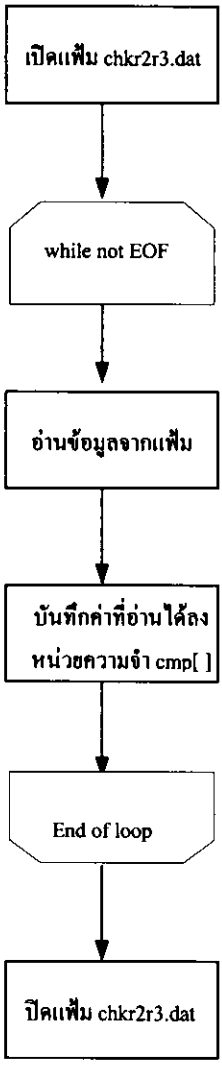
3.11 แผนผังระบบ (System Flow Chart)



ภาพประกอบที่ 3.24 แผนผังระบบการตรวจจับการบุกรุกในภาพรวม

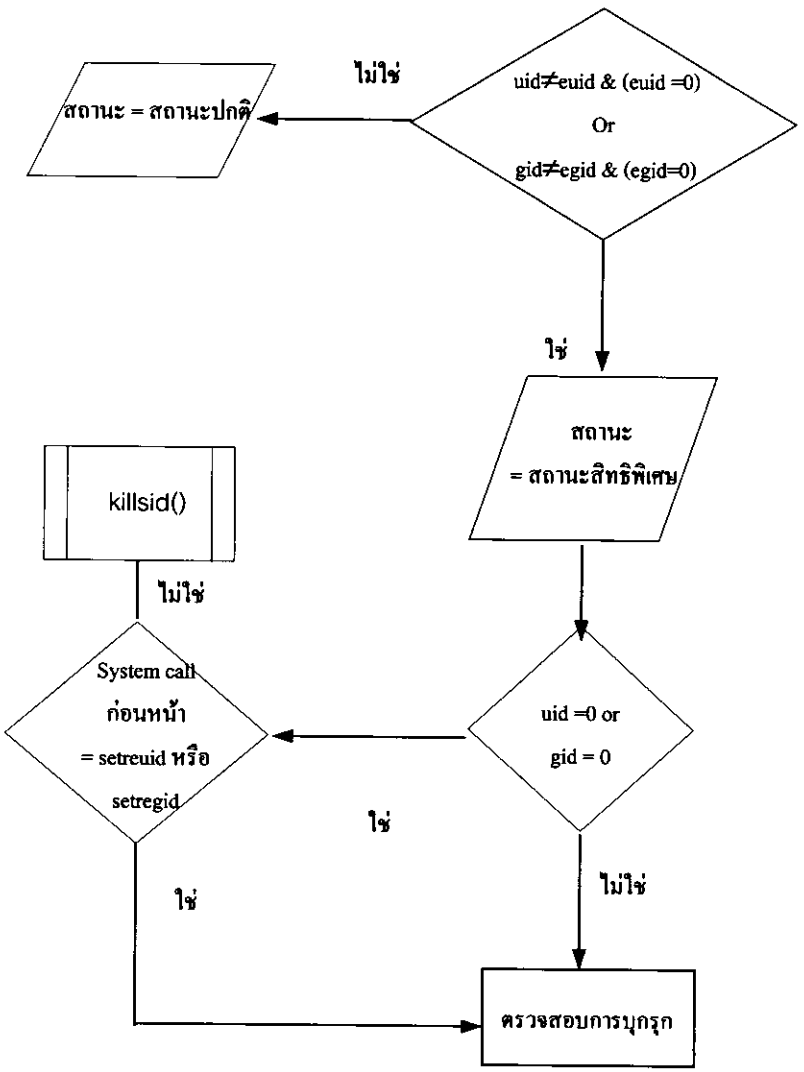
3.11.1 แผนผังระบบสำหรับฟังก์ชัน Load2Buff()

ฟังก์ชัน Load2Buff() ทำหน้าที่สำเนาข้อมูลจากแฟ้ม “chkr2r3.dat” ใส่ในหน่วยความจำ



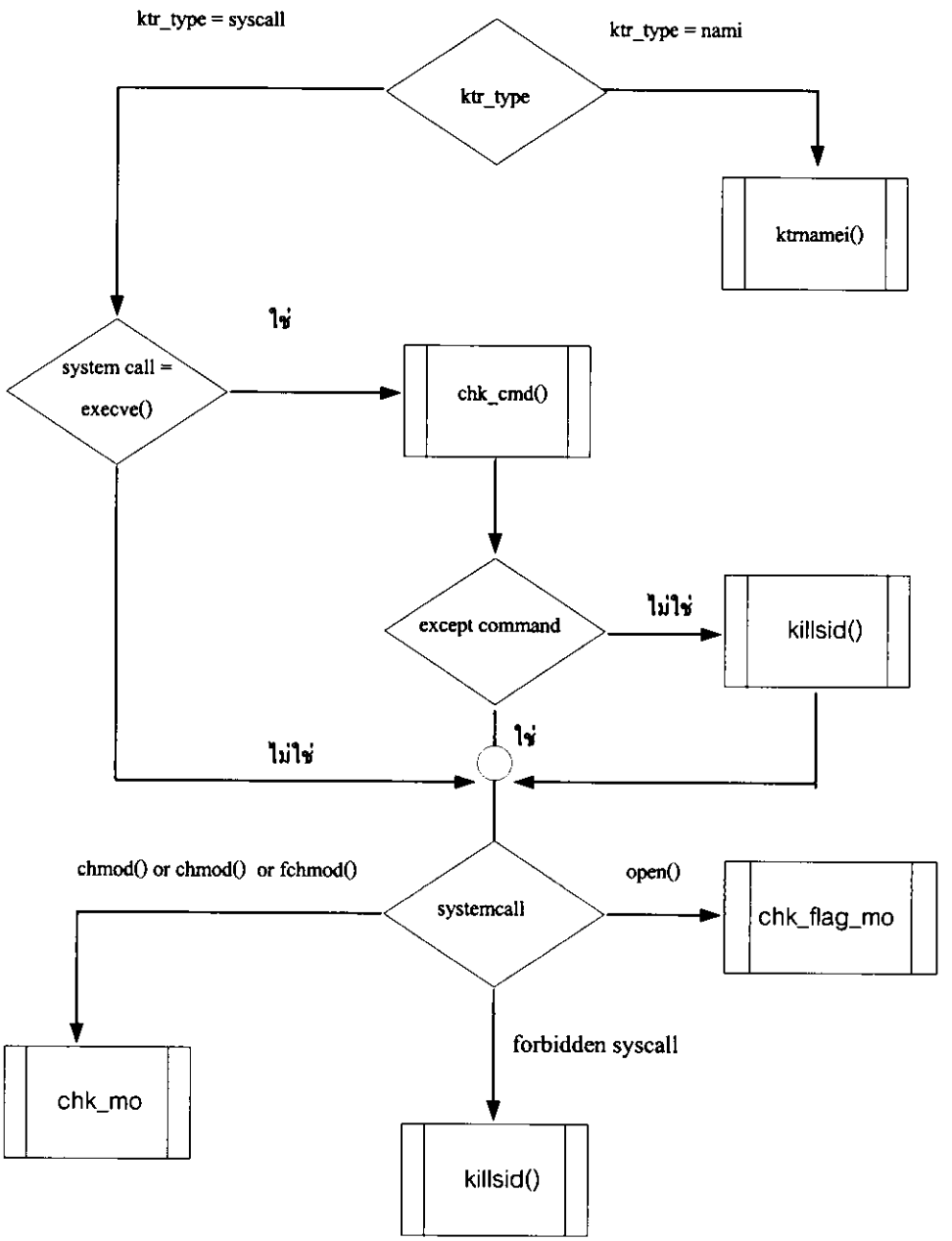
ภาพประกอบที่ 3.25 แผนผังระบบการทำงานของฟังก์ชัน Load2Buff()

3.11.2 แผนผังระบบการทำงานของการตรวจสอบสถานะ



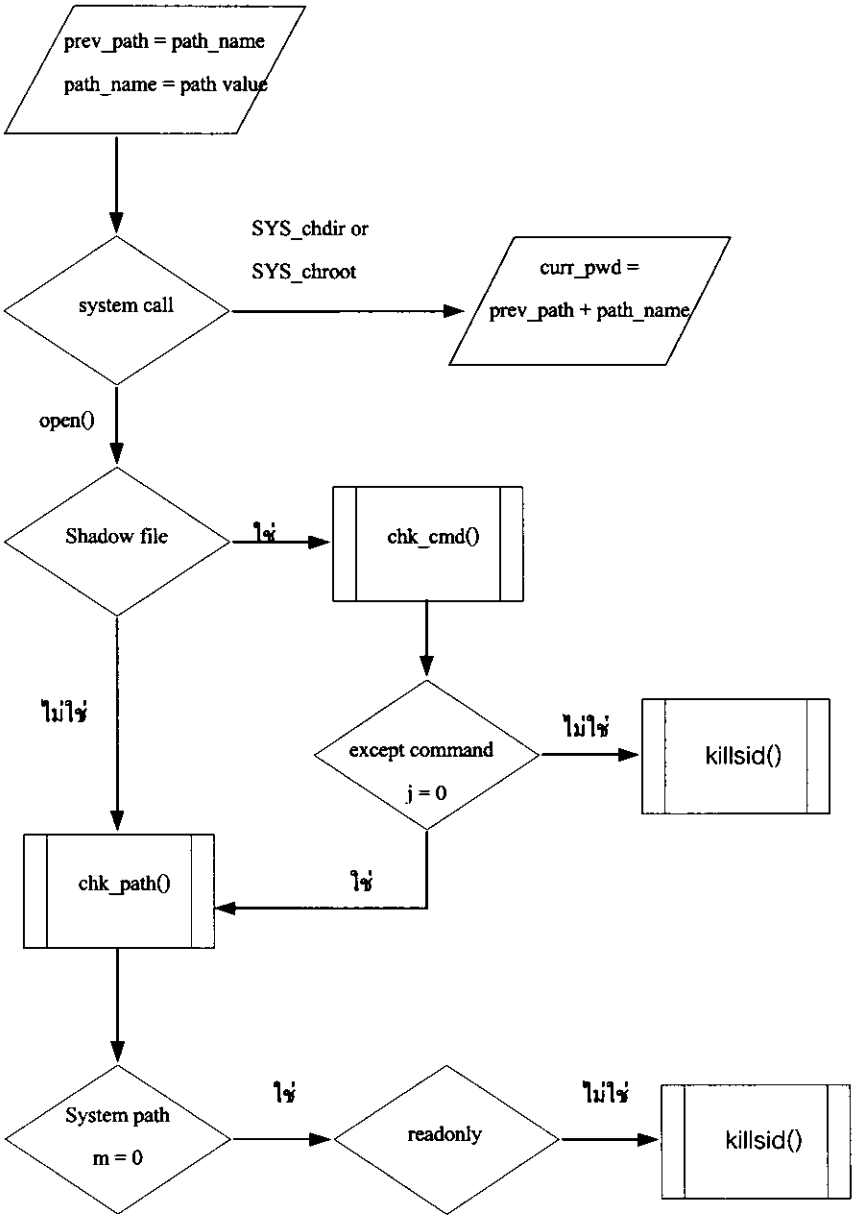
ภาพประกอบที่ 3.26 แสดงแผนผังระบบการทำงานของการตรวจสอบสถานะ

3.12 แผนผังระบบแสดงการทำงานของการตรวจสอบการบุกรุก



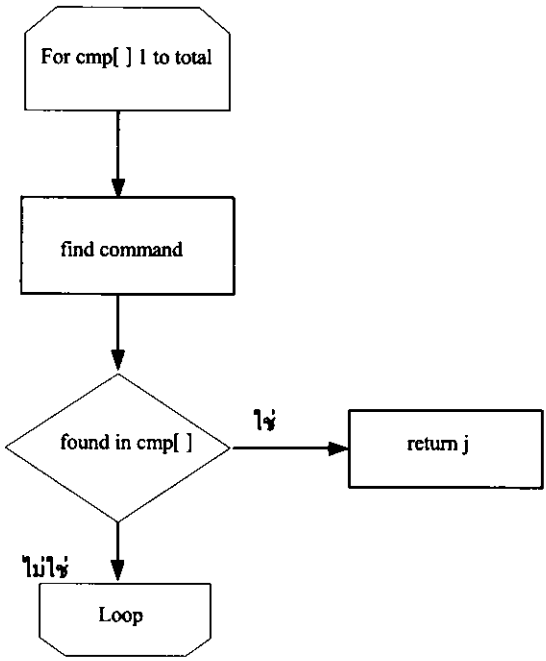
ภาพประกอบที่ 3.27 แสดงแผนผังระบบการทำงานของการตรวจสอบการบุกรุก

3.12.1 แผนผังระบบแสดงการทำงานฟังก์ชัน ktrnamei()



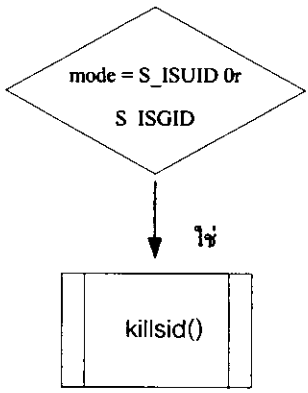
ภาพประกอบที่ 3.28 แสดงแผนผังระบบการทำงานของฟังก์ชัน ktrnamei()

3.12.2 แผนผังระบบแสดงการทำงานฟังก์ชัน chk_cmd()



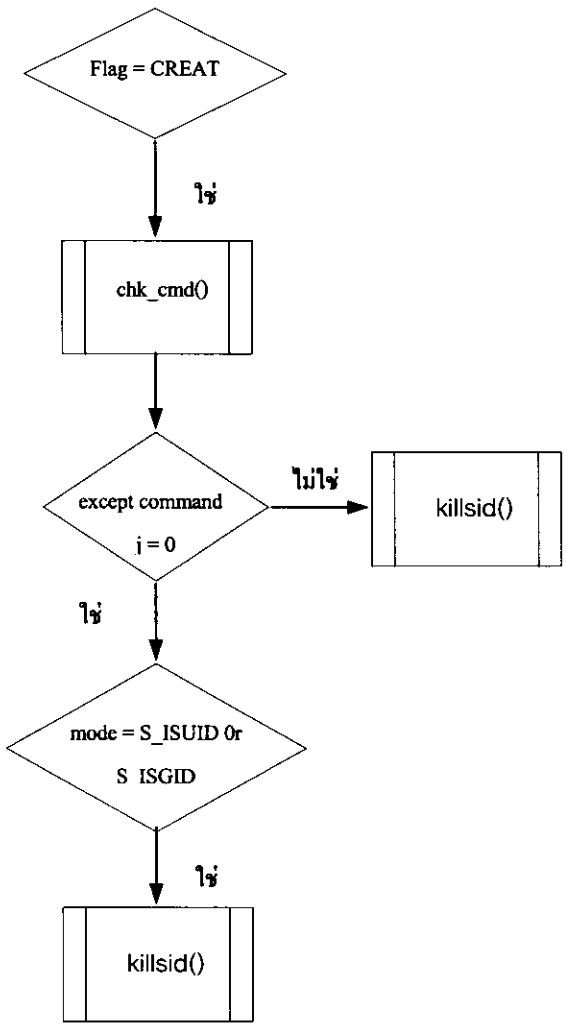
ภาพประกอบที่ 3.29 แสดงแผนผังระบบการทำงานของฟังก์ชัน chk_cmd()

3.12.3 แผนผังระบบแสดงการทำงานฟังก์ชัน chk_mode()



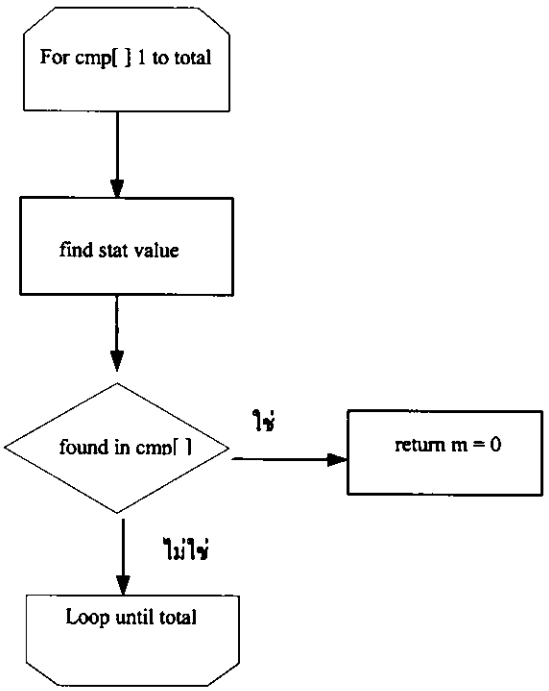
ภาพประกอบที่ 3.30 แสดงแผนผังระบบการทำงานของฟังก์ชัน chk_mode()

3.12.4 แผนผังระบบแสดงการทำงานฟังก์ชัน chk_flag_mode()



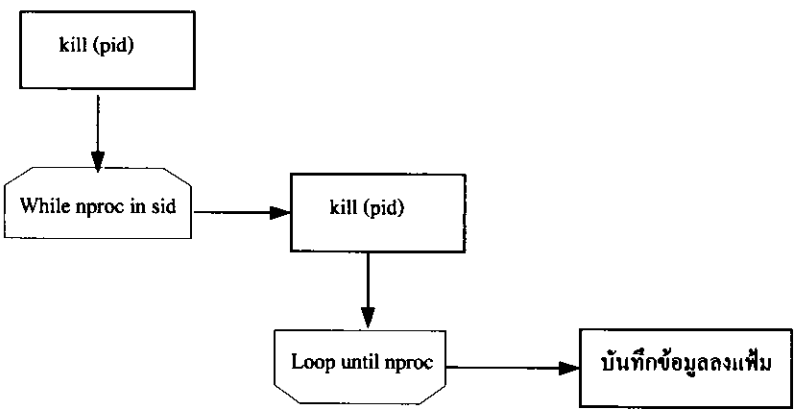
ภาพประกอบที่ 3.31 แสดงแผนผังระบบการทำงานของฟังก์ชัน chk_flag_mode()

3.12.5 แผนผังระบบแสดงการทำงานฟังก์ชัน chk_path()



ภาพประกอบที่ 3.32 แสดงแผนผังระบบการทำงานของฟังก์ชัน chk_path()

3.12.6 แผนผังระบบแสดงการทำงานฟังก์ชัน killsid()



ภาพประกอบที่ 3.33 แสดงแผนผังระบบการทำงานของฟังก์ชัน killsid()

3.13 การพัฒนาโปรแกรม

ในการพัฒนาระบบสำหรับวิทยานิพนธ์ชิ้นนี้ได้นำสิ่งที่ออกแบบดังที่กล่าวไว้ข้างต้นมาดำเนินการพัฒนาระบบตรวจจับการบุกรุกโดยพัฒนาขึ้นบนระบบปฏิบัติการ NetBSD เวอร์ชัน 1.6x และใช้ภาษาซีเป็นเครื่องมือในการพัฒนา ในลำดับแรกของการพัฒนาโปรแกรมได้ทำการแก้ไขแฟ้มต่าง ๆ ที่เกี่ยวข้องกับ system call *ktrace()* เพื่อให้ได้ข้อมูลเกี่ยวกับค่าประจำตัวผู้ใช้ (UID EUID GID EGID)ในการนำมาพิจารณาสถานะของโปรเซสในขณะนั้นแฟ้มที่แก้ไขได้แก่ /usr/src/sys/sys/ktrace.h และแฟ้ม /usr/src/sys/kern/kern_ktrace.c โดยเพิ่มเติมในส่วนของข้อมูลผู้ใช้ดังแสดงตามแฟ้มในภาคผนวก หลังจากนั้นทำการคอมไพล์เคอเนลใหม่ เพื่อใช้เป็นระบบปฏิบัติการสำหรับระบบตรวจจับการบุกรุกที่พัฒนาขึ้นมานี้ ในหัวข้อถัดไปจะกล่าวถึงลักษณะของระบบตรวจจับการบุกรุกที่พัฒนาขึ้น

3.13.1 ลักษณะของระบบตรวจจับการบุกรุกที่พัฒนาขึ้น

การทำงานของโปรแกรมจะเริ่มขึ้นหลังจากที่บริการที่ถูกติดตั้งบนระบบเริ่มทำงาน โดยที่ตัวระบบตรวจจับการบุกรุกจะทำงานอยู่เบื้องหลังและติดตามการทำงานของแต่ละบริการ (หรือโปรเซส) แบบรับช่วง (*inherit*) นั่นคือเมื่อบริการที่ถูกติดตามมีการแตกโปรเซสใหม่ ระบบตรวจจับการบุกรุกจะติดตามการทำงานของโปรเซสย่อยนั้นด้วยเช่นกัน จำนวนของโปรเซสที่ถูกติดตามในตอนเริ่มต้นจะขึ้นอยู่กับจำนวนบริการที่ถูกกำหนดให้ทำงานในขณะนั้น ซึ่งได้มาจากการเรียกใช้ฟังก์ชัน *kvm_getprocs()* ซึ่งจะส่งค่ากลับเป็นหมายเลขโปรเซสที่ทำงานอยู่บนระบบขณะนั้น แต่ละโปรเซสจะถูกติดตามด้วยฟังก์ชัน *idssysc()* ในลักษณะแบบรับช่วงดังที่กล่าวข้างต้น ซึ่งข้อมูลของโปรเซสที่ถูกติดตามจะถูกบันทึกไว้เป็นส่วนแรกของแฟ้มที่บันทึกการตรวจจับ *"ids.log"* เมื่อระบบตรวจจับการบุกรุกเริ่มทำงานดังตัวอย่างที่แสดงในภาพประกอบที่ 3.27 การทำงานของฟังก์ชัน *idssysc()* จะเริ่มจากการอ่านข้อมูลจากแฟ้ม *"chkr2r3.dat"* ซึ่งเป็นข้อมูลของแฟ้มระบบที่ถูกนิยามไว้รวมทั้งคำสั่งที่ถูกยกเว้นในการติดตามมาเก็บไว้ในหน่วยความจำ หลังจากนั้นระบบจึงเริ่มวิเคราะห์ข้อมูลเข้าที่ได้มาจากโปรเซส ในการทำงานของระบบเมื่อมีการเรียกใช้ system call *chdir()*, *chroot()* ค่าพารามิเตอร์ของ system call ทั้งสองซึ่งเป็นค่าของไดเรกทอรี จะถูกเก็บไว้ทุกครั้งเพื่อใช้ในการอ้างถึง ไดเรกทอรีปัจจุบันในการเปรียบเทียบค่าที่ชี้ตำแหน่งแฟ้มหรือไดเรกทอรี เมื่อมีการเปรียบเทียบการเรียกใช้แฟ้มชาโดว์ หรือการเปรียบเทียบไดเรกทอรีในการเรียกใช้แฟ้มระบบตามกฎข้อที่ 3 ในขณะเดียวกันเมื่อโปรเซสที่ถูกติดตามอยู่ใน

สถานะสิทธิพิเศษซึ่งพิจารณาจากการเปลี่ยนแปลงค่าของ EUID หรือ EGID system call ที่ถูกติดตามได้แก่ *open() fopen() chmod() lchmod() fchmod* โดยจะตรวจสอบจากแฟ้มที่ถูกเรียกใช้ค่าแฟลกหรือค่าของโหมด ตามการออกแบบกฎแต่ละข้อ นอกจากนี้ยังมี system call ตามที่ระบุในกฎที่ใช้สนับสนุนการตรวจจับข้อที่ 5 หากพบว่าโปรเซสใดที่เข้าข่ายการบุกรุก โปรเซสนั้นจะถูกหยุดการทำงานด้วย System call *kill()* ตามด้วยโปรเซสที่อยู่ในเซสชันเดียวกัน ในขั้นตอนสุดท้ายของระบบตรวจจับการบุกรุกข้อมูลที่ได้จากการติดตามจะถูกบันทึกลงในแฟ้ม *"ids.log"* ต่อจากข้อมูลโปรเซสที่ถูกตรวจจับดังกล่าวข้างต้นโดยมีรูปแบบดังนี้

```
10 Aug 26 17:48:25 tualek idssysc: trace from 1 Rule 0 syscall 181 session 97 pid 100 is killed
```

โดยที่

10 Aug 26 17:48:25	หมายถึง	วันที่และเวลาที่ตรวจจับได้
tualek	หมายถึง	ชื่อเครื่องที่ให้บริการ
idssyc	หมายถึง	โปรแกรมที่ตรวจจับ
trace from 1	หมายถึง	เป็นการตรวจจับโดยรับช่วงจากการติดตามการทำงานของโปรเซสที่มีหมายเลขโปรเซสเท่ากับ 1
Rule 0	หมายถึง	เป็นการตรวจจับตามเงื่อนไขกฎข้อที่ 1
syscall 181	หมายถึง system call 181	system call ที่ถูกเรียกใช้ขณะตรวจจับคือ
session 97	หมายถึง ว่าเป็นการบุกรุกคือ 97	หมายเลขเซสชันของโปรเซสที่ถูกตรวจจับ
pid 100	หมายถึง บุกรุกคือ 100	หมายเลขโปรเซสที่ถูกตรวจจับว่าเป็นการ

ในภาพประกอบที่ 3.34 แสดงตัวอย่างข้อมูลการตรวจจับที่ถูกบันทึกในแฟ้มบันทึกข้อมูลการบุกรุก

```
Aug 12 00:11:09 tualek idssysc:  my own pid sid 399 12
Aug 12 00:11:09 tualek idssysc: pid 372  ppid 1 pgid 372 sid 372 trcpid 408
Aug 12 00:11:09 tualek idssysc: pid 319  ppid 1 pgid 319 sid 319 trcpid 401
Aug 12 00:11:09 tualek idssysc: pid 342  ppid 1 pgid 342 sid 342 trcpid 405
Aug 12 00:11:09 tualek idssysc: pid 304  ppid 1 pgid 304 sid 304 trcpid 378
Aug 12 00:11:09 tualek idssysc: pid 188  ppid 1 pgid 188 sid 188 trcpid 409
Aug 12 00:11:09 tualek idssysc: pid 157  ppid 1 pgid 157 sid 157 trcpid 370
Aug 12 00:11:09 tualek idssysc: pid 12  ppid 1 pgid 12 sid 12 trcpid 414
Aug 12 00:11:09 tualek idssysc: pid 1 ppid 0 pgid 1 sid 1 trcpid 366
```

ภาพประกอบที่ 3.34 ตัวอย่างการบันทึกข้อมูลโปรเซสที่ถูกติดตามในแฟ้มบันทึกข้อมูลการบุกรุก

my own pid	หมายถึง	หมายเลขโปรเซสหลักของระบบ
pid	หมายถึง	หมายเลขโปรเซสที่ถูกติดตาม
ppid pgid sid	หมายถึง	โปรเซสแม่ กลุ่มโปรเซส และเซสชันของโปรเซสที่ถูกติดตาม
trcpid	หมายถึง	หมายเลขของโปรเซสที่ทำหน้าที่ติดตาม

3.13.2 ฟังก์ชันที่ถูกเรียกใช้ในระบบตรวจจับการบุกรุก

idstrace()

รูปแบบการใช้งาน

```
int idstrace(int pid, int tracepid)
```

คำอธิบาย

ฟังก์ชัน idstrace() จะรับค่าหมายเลขโปรเซสที่ถูกติดตาม pid และหมายเลขโปรเซสที่ติดตาม โดยที่จะมีการเรียกใช้ system call *ktrace()* ในการติดตามการทำงานของโปรเซสที่ถูกระบุด้วยหมายเลข pid

load2Buff()

รูปแบบการใช้งาน

int Load2Buff()

คำอธิบาย

เป็นฟังก์ชันที่ใช้สำหรับการอ่านข้อมูลจากแฟ้ม *"chkr2r3.dat"* มาเก็บไว้ในหน่วยความจำเพื่อใช้ตรวจสอบคำสั่งหรือที่อยู่ของแฟ้มตามกฎข้อที่ 2 และ 3

fread_tail()

รูปแบบการใช้งาน

int fread_tail((char *)&ktr_header, sizeof(struct ktr_header),1,fp)

คำอธิบาย

เป็นฟักชันที่ใช้ในการอ่านข้อมูลที่ได้จากการติดตามโดยใช้ system call ktrace() เข้ามาครั้งละหนึ่งเรคคอร์ดตามรูปแบบข้อมูลเข้าที่กล่าวไว้ในบทที่ 3

ktrsyscall()

รูปแบบการใช้งาน

void ktrsyscall ((struct ktr_syscall *));

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการตรวจสอบ system call ที่ถูกอ่านเข้ามา

chk_cmd()

รูปแบบการใช้งาน

int chk_cmd(char pcmd[])

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการตรวจสอบคำสั่งที่ได้รับการยกเว้น

chk_path()

รูปแบบการใช้งาน

int chk_path(char pcmd[], pid_t c_pid)

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการตรวจสอบตำแหน่งของแฟ้มเพื่อเปรียบเทียบกับค่าที่เก็บไว้ในหน่วยความจำ

chk_flg_mode()

รูปแบบการใช้งาน

int chk_flg_mode (int, int)

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการตรวจสอบแฟลกและโหมดของ system call *open()* ที่ใช้เพื่อสร้างแฟ้มใหม่

chk_mode()

รูปแบบการใช้งาน

chk_mode (int)

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการตรวจสอบแฟลกของ system call *chmod()*, *lchmod()* และ *fchmod()*

ktrsysret()

รูปแบบการใช้งาน

ktrsysret (struct ktr_sysret *, int)

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการตรวจสอบค่าที่ system call ที่ถูกเรียกใช้ส่งค่ากลับ

ktrnamei()

รูปแบบการใช้งาน

ktrnamei (char *, int)

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการรับค่าของตำแหน่งและชื่อของแฟ้ม

killsid()

รูปแบบการใช้งาน

killsid(pid_t,int rule,int ,pid_t,pid_t);

คำอธิบาย

เป็นฟังก์ชันที่ใช้ในการหยุดการทำงานของโปรเซสที่ถือว่าเป็นการกระทำที่เข้าข่ายการบุกรุก และบันทึกข้อมูลที่ได้จากการตรวจจับลงในแฟ้มบันทึกข้อมูลการบุกรุก

3.13.3 โค้ดเทียมแสดงการทำงานของระบบตรวจจับการบุกรุก

```
// Chk_State()
If process uid equal to owner uid = 0
        exit ()
Else
        Set curent process id to previous process id
End if
Do while process not terminate
        If process uid not equal process euid
                if process uid not equal the owner uid
                        if previous system call is setreuid
                                Set state = setreuid state
                                Do check_ids()
                        Else
                                Detected()
                        End if
                Else
                        Set state = setuid state
                        Do check_ids()
                End if
    End if
 If process gid not equalto  process egid
                if process gid not equal tothe owner gid
                        if previous System call is setregid
                                Set state = setregid state
                                Do check_ids()
                        Else
                                Detected()
                        End if
                Else
                        Set state = setgid state
                        Do check_ids()
               End if
    End if
End While

// Check_ids()
If System call = execve()
        If not in except_execve_group
                Do detected()
        Else
                Do Chk_State()
        End if
Else
        Do Chk_SysCall()
End if

// Chk_SysCall()
```

```
If System call in Stricted_SysCall_List
        Do detected()
Else
        If System call in Setuid_Setgid_List
                Do detected()
        Else
                If System call is Open()
                        Do Chk_argv()
                End if
        End if
End if
// Chk_argv()
If file in system_file_list and option try to modify these file
        Do detected()
End if
If file in create_new_user_list and option try to create new user
        Do detected()
End if
// Detected()
        write ids_log_file
        kill login process
```

3.14 สรุป

ในบทนี้ได้กล่าวถึงการวิเคราะห์ การออกแบบ และการพัฒนาโปรแกรมที่ใช้ในการตรวจจับการบุกรุก โดยเริ่มจากการวิเคราะห์ข้อมูลที่จะใช้เป็นข้อมูลเข้าสำหรับโปรแกรมการวิเคราะห์การบุกรุก กฎที่ใช้สนับสนุนการตรวจจับการบุกรุก ถัดมาเป็นการออกแบบโปรแกรมย่อยตามกฎแต่ละข้อ การออกแบบโปรแกรมในภาพรวม โดยในส่วนของการออกแบบระบบได้แสดงในรูปแบบของ DFD เพื่อให้เห็นถึงกระแสการไหลของข้อมูลในระบบ FSM เพื่อแสดงถึงกลไกสถานะระบบ และ System Flow Chart แสดงแผนผังการทำงานของระบบในแต่ละขั้นตอน ในส่วนของการพัฒนาได้กล่าวถึงขั้นตอนในการทำงานของโปรแกรม ฟังก์ชันที่ใช้ในโปรแกรม ในบทถัดไปจะกล่าวถึงการทดสอบโปรแกรมตรวจจับการบุกรุก