โปรแกรมซีเคียวเชลเซิร์ฟเวอร์สำหรับวินโดวส์

Secure Shell Server for Windows

โดย

นายธนาธิป ยินดี

นายปิติ จิโรจน์กุล

อาจารย์ที่ปรึกษา

อ.ธนา หงษ์สุวรรณ

อ.อักรเดช วัชระภูพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ ปีการศึกษา 2546

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมซีเคียวเชลเซิร์ฟเวอร์สำหรับวินโดวส์

Secure Shell Server for Windows

ผู้จัดทำ

1. นายธนาธิป ยินดี รหัสประจำตัว 43010167
2. นายปิติ จิโรจน์กุล รหัสประจำตัว 43010267

_____ อาจารย์ที่ปรึกษา

(อาจารย์ธนา หงษ์สุวรรณ)

_____ อาจารย์ที่ปรึกษา

(อาจารย์อัครเดช วัชระภูพงษ์)

โปรแกรมซีเคียวเชลเซิร์ฟเวอร์สำหรับวินโดวส์

นายธนาธิป ยินดี
นายปิติ จิโรจน์กุล
อาจารย์ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษา
อาจารย์อัครเดช วัชระภูพงษ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2546

บทคัดย่อ

ปัจจุบันมีผู้ใช้งานระบบปฏิบัติการวินโดวส์เพิ่มขึ้นเป็นจำนวนมาก โปรแกรมอย่างเทลเน็ต (Telnet) ก็มีปัญหาด้านความปลอดภัย เนื่องจากข้อมูลที่ส่งไปไม่มีการเข้ารหัสลับ จึงทำให้การดูแลรักษาและการใช้งานจากระยะไกลก็จะมีปัญหาด้านความปลอดภัยด้วย รวมทั้งระบบปฏิบัติการวินโดวส์ขาดโปรแกรมที่ทำหน้าที่เป็นเซิร์ฟเวอร์ที่สามารถรับการเชื่อมต่อจากระยะไกลได้อย่างปลอดภัย ดังนั้นปริญญานิพนธ์ฉบับนี้ จึงเสนอโปรแกรมที่สามารถลดปัญหาที่กล่าวมาได้คือโปรแกรมซีเคียวเชลเซิร์ฟเวอร์สำหรับวินโดวส์ ซึ่งมีความเข้ากันได้กับโพรโตคอลซีเคียวเชลเซิร์ฟเวอร์เวอร์ชันที่ 2 สนับสนุนการทำอินเทอร์แอกทีฟเซสชัน และการพิสูจน์ตนโดยใช้บัญชีผู้ใช้เดียวกับระบบปฏิบัติการ

Secure Shell Server for Windows

Thanatip Yindee

Bhiti Jirojanakul

Thana Hongsuwan Advisor

Akkradach Watcharapupong Advisor

ABSTRACT

Nowsaday, there is an increasing number of Windows users. Because of the unencrypted transmission in Telnet-like programs, some security problems occurred in maintenance and remote-access. Due to the lack of secure remote-access server program on Windows platform, this thesis offer Secure Shell Server for Windows as a solution. This program is compatible with Secure Shell Protocol version 2, provides interative-session and supports authentication via Windows account system.

กิตติกรรมประกาศ

โครงงานและปริญญานิพนธ์ฉบับนี้เสร็จสมบูรณ์ได้ เนื่องจากคำแนะนำจากอาจารย์ที่ปรึกษาทั้งสองท่านคือ อาจารย์ธนา หงษ์สุวรรณ และอาจารย์อัครเดช วัชระภูพงษ์ ที่คอยแนะนำ เป็นที่ปรึกษา และเอาใจใส่กับการทำโครงงานนี้เป็นอย่างดี ซึ่งทางคณะผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษาทั้งสองท่านเป็นอย่างสูง

นอกเหนือจากนี้ก็ต้องขอขอบพระคุณคณะอาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอย่างยิ่งที่ได้ช่วยประสิทธิ์ประสาทวิชาความรู้ให้แก่คณะผู้จัดทำ อีกทั้งภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้เอื้อเฟื้ออุปกรณ์ทรัพยากร สถานที่ และอำนวยความสะดวกต่าง ๆ ในการทำโครงงานนี้ด้วย

ขอขอบคุณเพื่อน ๆ พี่ ๆ และน้อง ๆ ชาว ISAG ได้แก่ พี่ต่อ พี่เดียร์ พี่เชาว์ พี่บอมบ์ โพสต์แมน ตั้ว เบส และคนอื่น ๆ ซึ่งได้ช่วยเหลือในการทำงานและแก้ไขปัญหา อุปสรรคต่าง ๆ ให้ผ่านพ้นไปด้วยดี

สุดท้ายนี้ต้องขอบพระคุณบุคคลที่สำคัญที่สุดคือ บิดา และมารดา ที่เคารพและเป็นที่รักยิ่ง ผู้ที่ให้กำเนิด คอยสั่งสอน ให้การศึกษาอย่างสูงสุด พร้อมทั้งสนับสนุนสิ่งต่าง ๆ นับเป็นพระคุณอย่างสูงสุดหาที่เปรียบมิได้ คณะผู้จัดทำขอระลึกพระคุณอันยิ่งใหญ่สุดประมาณนี้ไว้กว่าชีวิตจะหาไม่ และกราบขอบพระคุณทุกท่านไว้ ณ ที่นี้ด้วย

ธนาธิป ยินดี

ปิติ จิโรจน์กุล

สารบัญ

บทคัดย่อ	I
ABSTRACT	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญรูป	VII
บทที่ 1 บทนำ	**1**
1.1 ความสำคัญและที่มา	1
1.2 วัตถุประสงค์	1
1.3 ขอบเขตของงานวิจัย	2
1.4 ผลที่คาดว่าจะได้รับ	2
บทที่ 2 การเข้ารหัสลับ (Encryption) และแมสเซสไดเจส (Message Digest)	**3**
2.1 การเข้ารหัสลับแบบใช้กุญแจเดี่ยว	3
2.2 การเข้ารหัสลับแบบใช้กุญแจสาธารณะ	3
2.3 แมสเซสไดเจส	5
2.3.1 Secure Hash Algorithm	6
บทที่ 3 เทลเน็ต	**8**
3.1 เทลเน็ตทำงานอย่างไร	8
3.2 เทลเน็ตมีหลักการทำงานเบื้องต้นอย่างไร	9
3.3 Network Virtual Terminal (NVT)	10
3.3.1 โครงสร้างหลักและหลักการของ NVT	10
3.3.2 การทำงานและออปชันของ NVT ในเทลเน็ต	10
3.4 โหมดการส่งข้อมูลเทลเน็ต	12
3.4.1 โหมดการส่งข้อมูลทีละบรรทัด	12
3.4.2 โหมดการส่งข้อมูลแบบทีละตัวอักษร	13
3.5 เทอร์มินอลเสมือน (Virtual Terminal)	13

บทที่ 4 ซีเคียวเชลล์ **15**

4.1 ภาพรวมของโพรโตคอล 15

4.1.1 การเข้ารหัสลับ (Encryption) 16

4.1.2 ความถูกต้องของข้อมูล (Integrity) 16

4.1.3 การพิสูจน์ตน (Authentication) 16

4.1.4 การบีบอัดข้อมูล (Compression) 16

4.2 ขั้นตอนการติดต่อ 17

4.2.1 การตรวจสอบเวอร์ชัน 17

4.2.2 ซีเคียวเชลล์เวอร์ชันที่ 1 18

4.2.2.1 เซิร์ฟเวอร์และไคลเอนท์เปลี่ยนมาใช้ Binary Packet Protocol 18

4.2.2.2 เซิร์ฟเวอร์ส่งกุญแจและอัลกอริทึมที่สนับสนุน 19

4.2.2.3 ไคลเอนท์ส่งกุญแจเซสชัน 19

4.2.2.4 เริ่มต้นการเข้ารหัสลับ 19

4.2.2.5 การพิสูจน์ตนของไคลเอนท์ 19

4.2.3 ซีเคียวเชลล์เวอร์ชันที่ 2 20

4.2.3.1 การสร้างการเชื่อมต่อที่ปลอดภัย 22

4.2.3.1 การเรียกชื่ออัลกอริทึม 22

4.2.3.2 การแลกเปลี่ยนกุญแจเซสชันและกุญแจเซิร์ฟเวอร์ 22

4.2.3.3 Public Key Infrastructure (PKI) 22

4.2.3.4 การพิสูจน์ตน 23

4.2.3.5 การเปลี่ยนกุญแจเซสชัน 23

4.3 เปรียบเทียบข้อแตกต่างของซีเคียวเชลล์เวอร์ชันที่ 1 และ 2

บทที่ 5 การออกแบบ **24**

5.1 โปรแกรม netcat 24

5.2 ไลบรารี CryptLib 24

5.2.1 การคอมไพล์ CryptLib 24

5.2.2 การใช้งาน CryptLib กับ Microsoft Visual C++ 25

5.2.3 Initialisation 25

5.2.4 Return Codes 25

5.2.5 การสร้างเซสชันของ CryptLib 26

5.2.6 การสร้างกุญแจส่วนตัว และกุญแจสาธารณะ 26

5.2.7 การเก็บกุญแจส่วนตัว และกุญแจสาธารณะลงในดิสก์ 27

5.2.8 การรับและส่งข้อมูลผ่านทางเซสชันของ CryptLib 27

5.3 เอพีไอของวินโดวส์ 27
5.4 การทำงานของโปรแกรม IsagSSH2 Server 2546 27

บทที่ 6 การทดสอบและผลการทดสอบ 30
6.1 จุดประสงค์การทดสอบ 30
6.2 สภาพแวดล้อมและโปรแกรมที่ใช้ในการทดสอบ 30
6.2.1 โปรแกรมที่ใช้ในการทดสอบ 30
6.2.2 การติดตั้งและปรับแต่ง IsagSSH2 Server 2546 30
6.2.2.1 การสร้างโฮสต์คีย์ (Host Key) 30
6.2.2.2 การติดตั้งโปรแกรม 30
6.2.2.3 การปรับแต่งโปรแกรม IsagSSH2 Server 2546 31
6.2.2.4 การเรียกใช้โปรแกรม 31
6.3 วิธีทดสอบและผลการทดสอบ 31
6.3.1 การทดสอบการเข้ารหัสลับของโปรแกรม 31
6.3.2 การทดสอบการพิสูจน์ตนด้วยบัญชีผู้ใช้ 32
6.3.3 การทดสอบการเปลี่ยนทีซีพีพอร์ต 33

บทที่ 7 บทวิจารณ์และสรุป 35
7.1 บทวิจารณ์และสรุปโพรโตคอลซีเคียวเชลล์ 2 35
7.2 บทวิจารณ์และสรุปโปรแกรม IsagSSH2 Server 2546 35
7.3 ข้อจำกัดของโปรแกรม IsagSSH2 Server 2546 35
7.4 แนวทางการพัฒนาต่อในอนาคต 36

ภาคผนวก ก Microsoft Visual C++ 6.0 37
ภาคผนวก ข Ethereal 39
ภาคผนวก ค Network Virtual Terminal 42
ภาคผนวก ง ANSI/VT100 Terminal Control 44
ภาคผนวก SSH Protocol Architecture (SSH-ARCH) 49
บรรณานุกรม 79

สารบัญรูป

รูป 2-01	การเข้ารหัสลับแบบใช้กุญแจเดี่ยว	3
รูป 2-02	การเข้ารหัสลับแบบใช้กุญแจสาธารณะ	4
รูป 2-03	การทำลายเซ็นต์ดิจิตอล	4
รูป 2-04	การพิสูจน์ตนบนการเชื่อมต่อที่ไม่ปลอดภัยด้วยแมสเซสไดเจส	5
รูป 2-04	Secure Hash Algorithm	7
รูป 3-01	รูปแบบของข้อมูลเทลเน็ตที่ส่งในเครือข่าย	8
รูป 3-02	โครงสร้างของทีซีพีเซ็กเมนต์	9
รูป 3-03	โครงสร้างของไอพีดาต้าแกรม	9
รูป 3-04	โครงสร้างการทำ NVT	11
รูป 3-05	โหมดการส่งข้อมูลทีละบรรทัด	12
รูป 3-06	โหมดการส่งข้อมูลทีละตัวอักษร	13
รูป 4-01	สถาปัตยกรรมของซีเคียวเชล	15
รูป 4-02	สถาปัตยกรรมของซีเคียวเชลเวอร์ชันที่ 1	16
รูป 4-03	Binary Packet Protocol	16
รูป 4-04	สถาปัตยกรรมของซีเคียวเชลเวอร์ชันที่ 2	17
รูป 4-05	ความสัมพันธ์ของโพรโตคอลซีเคียวเชลเวอร์ชันที่ 2	18
รูป 4-06	แสดงการสร้างการเชื่อมต่อที่ปลอดภัย	22
รูป 4-07	ความสัมพันธ์ของโพรโตคอลซีเคียวเชลเวอร์ชันที่ 2	23
รูป 5-01	เพิ่มไฟล์ cl32.lib ในโปรเจค	25
รูป 5-02	การเรียกใช้งาน cryptInit() และ cryptEnd()	25
รูป 5-03	การตรวจสอบค่าผิดพลาดจากค่าที่คืนจากฟังก์ชัน	26
รูป 5-04	การสร้างกุญแจส่วนตัว และกุญแจสาธารณะ	26
รูป 5-05	การเก็บกุญแจส่วนตัว และกุญแจสาธารณะลงในดิสก์	27
รูป 5-06	โครงสร้างของ IsagSSH2 Server 2546	28
รูป 6-1	ผลทดสอบการเข้ารหัสลับ	32
รูป 6-02	ผลการทดสอบบัญชีผู้ใช้ปกติ	32
รูป 6-03	ผลการทดสอบบัญชีผู้ใช้ที่ป้อนรหัสผ่านผิด ถูกยกเลิก หรือไม่มีอยู่	33

รูป 6-04	ข้อความผิดพลาด เนื่องจากมีโปรแกรมอื่นใช้พอร์ตอยู่	34
รูป 6-04	ข้อความผิดพลาด เนื่องจากมีพอร์ตอยู่นอกช่วงที่เป็นไปได้	34
รูป ข-01	โปรแกรม Ethereal	39
รูป ข-02	กรอบสนทนาการดักจับแพ็กเก็ต	40
รูป ข-03	ตัวอย่างเงื่อนไขที่ใช้กรองแพ็กเก็ต	40
รูป ข-04	ตัวอย่างผลลัพธ์จากการใช้เงื่อนไขกรองแพ็กเก็ต	41
รูป ข-05	Follow TCP Stream	41

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบันเครือข่ายคอมพิวเตอร์ได้มีบทบาทสำคัญต่อการใช้งานในชีวิตประจำวัน เป็นอย่างมาก ดังนั้นประเด็นเรื่องความปลอดภัยเป็นสิ่งที่ต้องคำนึงถึง โปรแกรมและโพรโตคอลที่ใช้งานในปัจจุบันมักผ่านการออกแบบมาตั้งแต่ช่วงที่ระบบเครือข่ายไม่ได้ใช้งานแพร่หลายและมีความสำคัญเหมือนเช่นในปัจจุบัน ซึ่งในขณะออกแบบไม่ได้คำนึงถึงเรื่องความปลอดภัยมากนัก ซึ่งส่งผลให้การใช้งานที่ต้องพึ่งพาโปรแกรมเหล่านั้น มีความเสี่ยงในประเด็นเรื่องความปลอดภัยได้

เทลเน็ตเป็นโปรแกรมหนึ่งที่ใช้งานกันแพร่หลาย แต่มีปัญหาในเรื่องความปลอดภัย เพราะข้อมูลที่ส่งเป็นข้อความตัวอักษรธรรมดา การดักจับ หรือแก้ไขสามารถทำได้ง่าย นอกจากเทลเน็ตยังมีโปรแกรมหรือโพรโตคอลอื่นๆ อีกมากที่มีปัญหาเช่นนี้ เช่น เอฟทีพี (FTP), หรือแม้แต่โปรแกรมพวกส่งข้อความด่วน (Instant Messaging) ก็มีปัญหาเช่นเดียวกัน

ซีเคียวเชล (Secure Shell : SSH) เป็นโพรโตคอลหนึ่งที่ออกแบบมาเพื่อใช้แทนที่หรือใช้ร่วมกับโปรแกรมที่มีปัญหาในด้านความปลอดภัย สนับสนุนการทำอินเทอร์แอกทีฟเซสชัน (interactive session) ทีซีพีพอร์ตฟอร์เวิร์ด (TCP port forwarding) และการแลกเปลี่ยนไฟล์อย่างปลอดภัย (SFTP) ซึ่งปัจจุบันมีการพัฒนาเป็นเวอร์ชั่นที่ 2 แล้ว และก็ได้รับความนิยมอย่างมาก แต่ปริมาณการใช้งานจำกัดอยู่เฉพาะในกลุ่มผู้ใช้ยูนิกซ์ และลินุกซ์ เพราะในวินโดวส์มีเฉพาะที่เป็นไคล์เอนท์ ไม่มีโปรแกรมที่ทำหน้าที่เป็นเซิร์ฟเวอร์

สำหรับเหตุผลที่มีโครงงานนี้สนับสนุนเฉพาะซีเคียวเชลเวอร์ชันที่ 2 อันเนื่องมาจากข้อด้อยต่าง ๆ ที่เกิดขึ้นกับซีเคียวเชลเวอร์ชันที่ 1 รวมทั้งโปรแกรมไคลเอนท์ที่ใช้กันในปัจจุบันต่างพากันสนับสนุนซีเคียวเชสเวอร์ชันที่ 2 กันหมด จึงไม่มีความจำเป็นที่จะต้องทำให้เซิร์ฟเวอร์สนับสนุนซีเคียวเชลเวอร์ชันที่ 1 อีกต่อไป

1.2 วัตถุประสงค์

1. เพื่อสร้างโปรแกรมซีเคียวเชลเซิร์ฟเวอร์ต้นแบบ ที่ทำงานภายใต้ระบบปฏิบัติการวินโดวส์ 2000

2. เพื่อศึกษาและพัฒนาการเขียนโปรแกรมทางด้านเครือข่ายและความปลอดภัยบนระบบปฏิบัติการวินโดวส์ 2000

1.3 ขอบเขตของงานวิจัย

1. สร้างโปรแกรมเซิร์ฟเวอร์ที่เรียกใช้งานอินเทอร์แอกทีฟเซสชัน
2. สร้างโปรแกรมเซิร์ฟเวอร์ที่เข้ากันได้กับโพรโตคอลซีเคียวเชล เวอร์ชันที่ 2
3. พัฒนาด้วยภาษา C/C++ โดยใช้ Microsoft Visual C++ 6.0 เป็นเครื่องมือในการพัฒนา
4. สามารถทำงานได้ ภายใต้ระบบปฏิบัติการวินโดวส์ 2000
5. มีการพิสูจน์ตนด้วยรหัสผ่าน โดยใช้บัญชีผู้ใช้เดียวกับระบบปฏิบัติการ

1.4 ผลที่คาดว่าจะได้รับ

1. โปรแกรมซีเคียวเชลเซิร์ฟเวอร์ต้นแบบ ที่ทำงานภายใต้ระบบปฏิบัติการณ์วินโดวส์ 2000
2. ได้รับความรู้ในการเขียนโปรแกรมที่ทำงานกับเครือข่าย
3. ได้รับความรู้ในวิธีการสื่อสารอย่างปลอดภัย

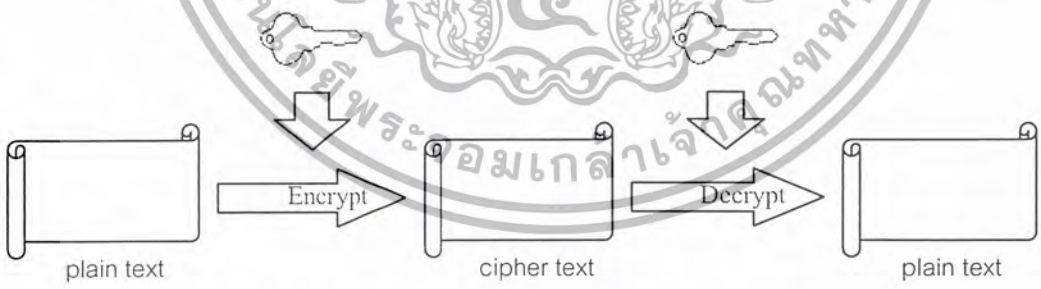
บทที่ 2
การเข้ารหัสลับและแมสเซสไดเจส

เมื่อการสื่อสารเข้ามามีบทบาทสำคัญในชีวิตประจำวัน ดังนั้นความจำเป็นที่จะสื่อสารกันอย่างปลอดภัยจึงมีมากขึ้น วิธีง่ายที่สุดในการติดต่อสื่อสารให้ปลอดภัยคือ การเข้ารหัสลับ (encryption) เพื่อไม่ให้ผู้ที่ไม่เกี่ยวข้องทราบข้อมูลเดิม การเข้ารหัสลับมีอยู่สองประเภทคือ การเข้ารหัสลับแบบใช้กุญแจเดี่ยว และการเข้ารหัสลับแบบใช้กุญแจสาธารณะ นอกจากนี้จำเป็นต้องตรวจสอบได้หากข้อมูลมีการเปลี่ยนแปลง ซึ่งในหัวข้อนี้จะกล่าวถึงแฮชฟังก์ชัน (hash function) หรือแมสเซสไดเจส (message digest)

2.1 การเข้ารหัสลับแบบใช้กุญแจเดี่ยว

การเข้ารหัสลับแบบใช้กุญแจเดี่ยว (Secret Key Cryptography) เป็นการเข้ารหัสลับแบบที่ผู้เข้ารหัสลับและผู้ถอดรหัสลับต้องทราบกุญแจหรือความลับชุดเดียวกัน ซึ่งหากกุญแจไม่เหมือนกันจะไม่สามารถถอดรหัสลับได้ หรือหากถอดได้ข้อมูลก็จะผิดไปจากเดิม ข้อมูลที่ไม่ได้เข้ารหัสลับจะเรียกว่า เพลนเท็กซ์ (plain text) และข้อมูลที่ถูกเข้ารหัสลับจะเรียกว่า ไซเฟอร์เท็กซ์ (cipher text) แสดงดังรูป 2-01 สำหรับอัลกอริทึม (algorithm) เข้ารหัสลับแบบใช้กุญแจเดี่ยวที่เป็นที่รู้จักกันก็จะมี DES, 3DES, IDEA และ AES เป็นต้น

ข้อดีของวิธีนี้คือทำงานได้เร็ว และสามารถเลือกกุญแจได้เอง แต่จะมีข้อด้อยเมื่อนำมาใช้กับระบบที่มีผู้ใช้จำนวนมาก เพราะแต่ละคู่ของการติดต่อต้องมีกุญแจเฉพาะคู่ ซึ่งหากมีคู่ของการติดต่อจำนวนมาก กุญแจก็จะมากตาม และจะทำให้การจัดการทำได้ยาก

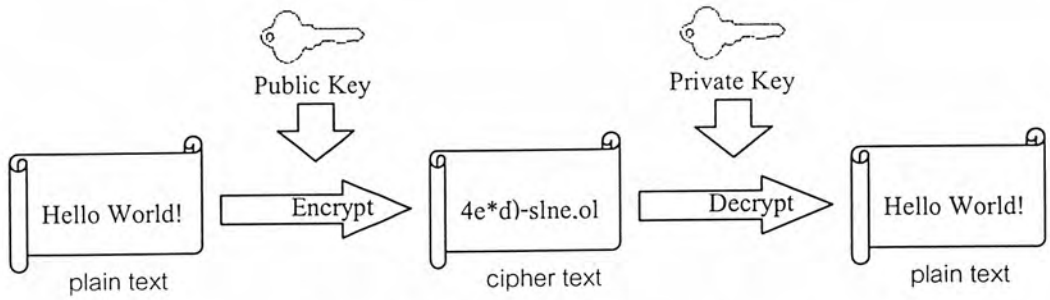


รูป 2-01 การเข้ารหัสลับแบบใช้กุญแจเดี่ยว

2.2 การเข้ารหัสลับแบบใช้กุญแจสาธารณะ

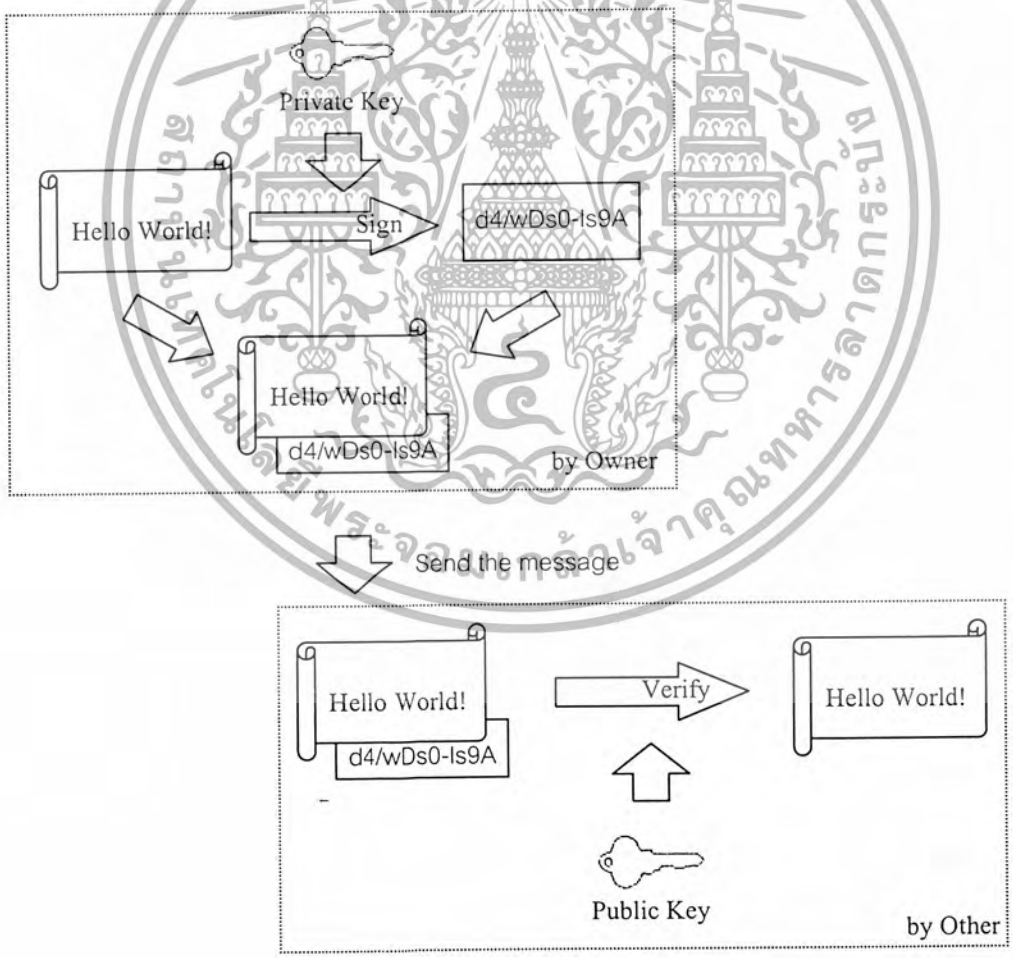
จากรูป 2-02 เป็นการเข้ารหัสลับแบบใช้กุญแจสาธารณะ (Public Key Cryptography) จะต่างกับการเข้ารหัสลับแบบใช้กุญแจเดี่ยวตรงที่ ในแต่ละบุคคล จะมีกุญแจอยู่สองดอก ดอกแรกใช้เข้ารหัสลับเรียกว่า กุญแจสาธารณะ (public key) อีกดอกใช้ถอดรหัสลับเรียกว่า กุญแจส่วนตัว (private key) โดยกุญแจสาธารณะจะประกาศให้ผู้อื่นทราบ และเมื่อผู้อื่นต้องการส่งข้อมูลมาหาเราให้ใช้ กุญแจสาธารณะเข้ารหัสลับข้อมูลแล้วส่ง จากนั้นให้ใช้กุญแจส่วนตัว ซึ่งเก็บไว้เป็นความลับห้ามบอกผู้อื่น ถอดรหัสลับ

ข้อมูลที่ส่งมา ด้วยวิธีนี้ผู้อื่นจะไม่สามารถถอดรหัสลับได้เลยหากไม่ทราบกุญแจส่วนตัว อัลกอริทึมที่ใช้วิธีนี้เช่น RSA และ Diffie-Hellman เป็นต้น



รูป 2-02 การเข้ารหัสลับแบบใช้กุญแจสาธารณะ

วิธีนี้มีข้อดีตรงที่ลดจำนวนกุญแจของแต่ละการติดต่อได้ เมื่อเทียบกับการเข้ารหัสลับแบบใช้กุญแจเดี่ยว จึงจัดการได้ง่ายกว่า แต่ก็มีข้อด้อยเช่นกัน คือ ทำงานได้ช้ากว่าเมื่อเทียบที่ความยาวของกุญแจเท่ากัน และไม่สามารถเลือกกุญแจเองได้



รูป 2-03 การทำลายเซ็นต์ดิจิตอล

นอกจากนี้การเข้ารหัสลับแบบใช้กุญแจสาธารณะ สามารถประยุกต์ใช้เป็นลายเซ็นดิจิตอล (digital signature) ได้ด้วย โดยผู้ส่งเซ็น (sign) ข้อความด้วยกุญแจส่วนตัว แล้วส่งข้อความนี้ไปให้ผู้รับ จากนั้นผู้รับจะตรวจสอบ (verify) ข้อความและลายเซ็นด้วยกุญแจสาธารณะของผู้ส่ง ซึ่งวิธีนี้สามารถยืนยันตัวบุคคลได้ว่าเป็นเจ้าของข้อความจริง ดังรูป 2-03

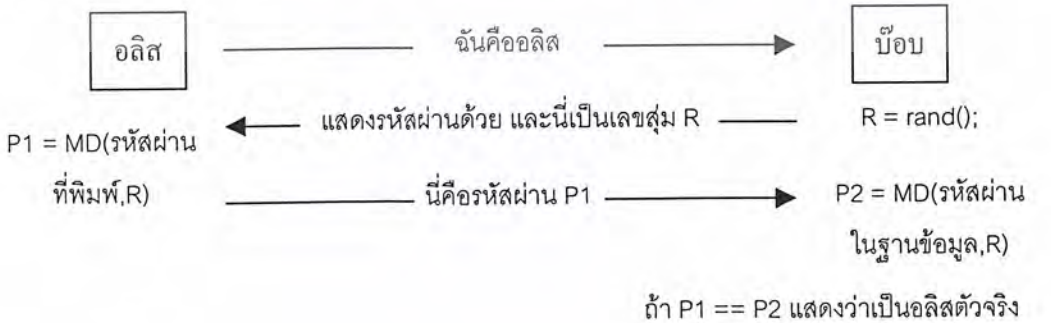
2.3 แมสเซสไดเจส (Message Digest)

นอกเหนือจากข้อมูลที่ส่งต้องเป็นความลับแล้ว อีกสิ่งหนึ่งที่สำคัญไม่แพ้กันคือข้อมูลชิ้นนั้นจะต้องเป็นข้อมูลเดิม ไม่มีการแต่งเติมเสริมต่อ ซึ่งอาจใช้วิธีพื้นฐานอย่างพาริตีบิต (parity bit) หรือ CRC แต่วิธีที่กล่าวมานี้ สามารถป้องกันความผิดพลาดได้เฉพาะความผิดพลาดที่เกิดด้วยความไม่ตั้งใจของมนุษย์ เช่น จากสัญญาณรบกวน หรือโปรแกรมที่ทำงานผิดพลาด แต่ไม่สามารถป้องกันความผิดพลาดที่เกิดจากความตั้งใจของมนุษย์ได้ ดังนั้นจึงต้องหาวิธีอื่นมาใช้งาน ซึ่งแมสเซสไดเจสสามารถแก้ปัญหานี้ได้

แมสเซสไดเจส หรือแฮส (hash) เป็นฟังก์ชันทางเดียว ซึ่งจะรับข้อความและตอบออกมาเป็นผลลัพธ์ที่ต่างๆ กัน โดยผลลัพธ์ที่ได้จะไม่สามารถทำย้อนกลับเป็นข้อความเดิมได้ในทางปฏิบัติ จึงเรียกว่าฟังก์ชันทางเดียว ฟังก์ชันแมสเซสไดเจสถือว่ามีความปลอดภัย เพราะว่าหากทราบแมสเซสไดเจสจะไม่มีทางคำนวณหาข้อความเดิม ได้ด้วยความสามารถในการคำนวณในปัจจุบัน ดังนั้นจึงเป็นไปไม่ได้ที่จะหาข้อความสองอันที่จะมีแมสเซสไดเจสเหมือนกัน

ฟังก์ชันแมสเซสไดเจสสามารถนำมาใช้ในการตรวจสอบความถูกต้องของข้อมูล (Integrity Checking) และมีข้อดีเหนือกว่าการใช้พาริตีบิต หรือ CRC เพราะฟังก์ชันแมสเซสไดเจสมีคุณสมบัติที่เรียกว่า strong avalanche คือ หากมีการเปลี่ยนแปลงของข้อความจากเดิมเพียงบิตเดียว แมสเซสไดเจสต้องมีการเปลี่ยนแปลงอย่างมากจนเห็นได้ชัด รวมทั้งฟังก์ชันแมสเซสไดเจสที่ใช้กันทั่วไปมีความยาวของแมสเซสไดเจสมากกว่า CRC มาก

นอกจากนี้ฟังก์ชันแมสเซสไดเจสอาจใช้ในการส่งรหัสผ่านในการพิสูจน์ตนบนการเชื่อมต่อที่ไม่ปลอดภัยโดยนำรหัสผ่านมาต่อกับตัวเลขสุ่มที่ได้รับจากอีกฝั่ง และผ่านฟังก์ชันแมสเซสไดเจส จากนั้นจึงส่งแมสเซสไดเจสกลับไป ผู้ที่อยู่ตรงกลางดักแมสเซสไดเจสได้ แต่ก็ไม่มีทางทราบว่ารหัสผ่านคืออะไร ดังรูป 2-04



รูป 2-04 การพิสูจน์ตนบนการเชื่อมต่อที่ไม่ปลอดภัยด้วยแมสเซสไดเจส

2.3.1 Secure Hash Algorithm

เพื่อให้เป็นมาตรฐานเดียวกัน NIST จัดมาตรฐานแฮชฟังก์ชัน เรียกว่า Secure Hash Algorithm (SHA) ถูกออกแบบโดยตัวแทนด้านความปลอดภัยแห่งชาติของสหรัฐอเมริกา (National Security Agency : NSA) เพื่อที่จะทำงานกับวิธีลายมือชื่อดิจิตอล วิธีนี้ให้อินพุทมีความยาวน้อยกว่า 2^{64} บิตซึ่งจะถูกลดรูปให้เหลือแมสเซสไดเจสเพียง 160 บิต

สัญลักษณ์นี้ $\oplus$ หมายถึงเอ็กซ์คลูซีฟออร์ (exclusive or) $\vee$ หมายถึงออร์ (or) $\wedge$ หมายถึงแอนด์ (and) และ $\neg$ หมายถึงนอต (not) โดยให้ฟังก์ชัน S(n,v) หมายถึง v เลื่อนไปทางซ้าย n ตำแหน่งเป็นวงกลม ทั้งหมดหารเศษด้วย 2^{32} (mod 2^{32})

ขั้นแรกข้อความจะถูกต่อด้วยเลข 1 หนึ่งตัว, 0 หลายๆตัว ให้เต็ม และ ค่า 64 บิตสำหรับจำนวนบิตในข้อความเดิม เพื่อว่าความยาวรวมของข้อความ ความยาวเลข 1 และ 0 หลายๆตัวและค่า 64 บิตเป็นเท่าของ 512 บิต

ข้อความจะถูกทำในบล็อกของเวิร์ด 16 ตัว(32-bits words จำนวน 16 ตัว) แสดงได้ดังนี้ W(0), W (1), W(2),...W(15) โดยที่ W(0) เป็นซ้ายสุด แต่ละบล็อกถูกขยายให้เป็นเวิร์ด 80 ตัว โดย W(t) := W(t-3) $\oplus$ W(t-8) $\oplus$ W(t-14) $\oplus$ W(t-16) สำหรับ t :=16 ถึง 79 จะได้ค่าคงที่ 5 ตัวคือ H0 := 67452301, H1 := EFCDAB89, H2 := 98BADCFE, H3 := 10325476 และ H4 := C3D2E1F0 (แสดงด้วยเลขฐาน 16) โดยมีวิธีแสดงดังรูป 6.1

หลังจากทำเสร็จสมบูรณ์แล้ว จะแมสเซสไดเจสขนาด 160 บิตจะมี เวิร์ด 5 ตัวคือ H0 H1 H2 H3 H4

วิธีนี้แสดงความกระจาย กระบวนการของการกระจายผลกระทบของบิตของเพลนเท็กซ์ตลอดทั้งไซเฟอร์เท็กซ์ ตัวอย่างเช่นในการขยายเริ่มแรกของเวิร์ด 16 ตัวไปเป็น 80 ตัว การเปลี่ยนแปลงในเวิร์ดที่ 14 มีผลโดยตรงกับเวิร์ดที่ 17, 22, 28 และ 30 แต่การเปลี่ยนในเวิร์ดที่ 17 จะมีผลกับเวิร์ดที่ 20, 25, 31 และ 33 การเปลี่ยนแปลงในเวิร์ดที่ 22 มีผลกับเวิร์ดที่ 25, 30, 36 และ 41 และไปเรื่อยๆ บิตบิตเดียวมีการเปลี่ยนแปลงในเวิร์ด 14 มีผลต่อเวิร์ด 17, 20, 22, 25, 28, 30, 31, 33, 34, 36, 37, 39, 41, 42, 44, 45, 47, 47, 49 และต่อไปเรื่อยๆ หลังจาก 80 ขั้นตอนแต่ละ W(J) จะมีผลต่อ TEMP ซึ่งจะมีผลต่อ A, B, C, D และ E ซึ่งจะส่งผลต่อ H0 ถึง H4 แต่ละตัว และจะออกมาเป็นผลลัพธ์ขั้นสุดท้าย

สรุปแล้วแฮชฟังก์ชันเป็นฟังก์ชันทางเดียวสามารถคำนวณหาผลลัพธ์ได้ง่ายแต่การคำนวณย้อนกลับทำได้ยาก เราสามารถนำประโยชน์ของแฮชฟังก์ชันมาใช้ในวิธีการทำลายมือชื่อดิจิตอล โดยใช้ในขั้นตอนของการทำแมสเซจไดเจส ถ้า f เป็นแฮชฟังก์ชันทางเดียว (one-way hash function) แล้ว f จะมีลักษณะดังต่อไปนี้

1) สามารถคำนวณหาค่า f(x) โดยค่าของ x มีขนาดเท่าใดก็ได้แต่จะได้ค่า f(x) ที่มีขนาดคงที่ ค่าของ f(x) ที่ได้ออกมา จะถูกเรียกว่าเป็นแฮชแวลู ซึ่งผลที่ได้จากการทำแฮชฟังก์ชันที่ต่างกันจะได้ผลลัพธ์ออกมามีขนาดที่ไม่เท่ากัน แต่ถ้าเลือกใช้ฟังก์ชันเดียวกันผลที่ได้ออกมาจะมีขนาดเท่ากัน
2) เมื่อกำหนดค่า x จะสามารถหาค่า f(x) ได้ง่าย

3) เมื่อกำหนดค่า y ซึ่ง y = f(x) การคำนวณหาค่า x จะทำได้ยากมาก

4) สำหรับค่า x_1 และ x_2 ใดๆ ที่ x_1 ไม่เท่ากับ x_2 จะไม่สามารถหาค่าที่ทำให้ $f(x_1) = f(x_2)$ ได้

```
for each 16-word block begin
      A := H0; B := H1; C := H2; D := H3; E := H4
      for I := 0 to 19 begin
            TEMP := S(5,A) + ((B∧C) ∨ (¬B∧D)) + E + W(I) + ↵
5A827999;
            E := D; D := C; C := S(30,B); B := A; A := TEMP
      end
      for I := 20 to 39 begin
            TEMP := S(5,A) + ((B⊕ C⊕D) + E + W(I) + 6ED9EBA1;
            E := D; D := C; C := S(30,B); B := A; A := TEMP
      end
      for I := 40 to 59 begin
            TEMP := S(5,A) + ((B∧C) ∨ (B∧D) ∨ (C∧D)) + E + ↵
W(I) + 8F1BBCDC;
            E := D; D := C; C := S(30,B); B := A; A := TEMP
      end
      for I := 60 to 79 begin
            TEMP := S(5,A) + ((B⊕ C⊕D) + E + W(I) + CA62C1D6;
            E := D; D := C; C := S(30,B); B := A; A := TEMP
      end
      H0 := H0+A; H1 := H1+B; H2 := H2+C; H3 := H3+ D; H4 := ↵
H4+E
end
```

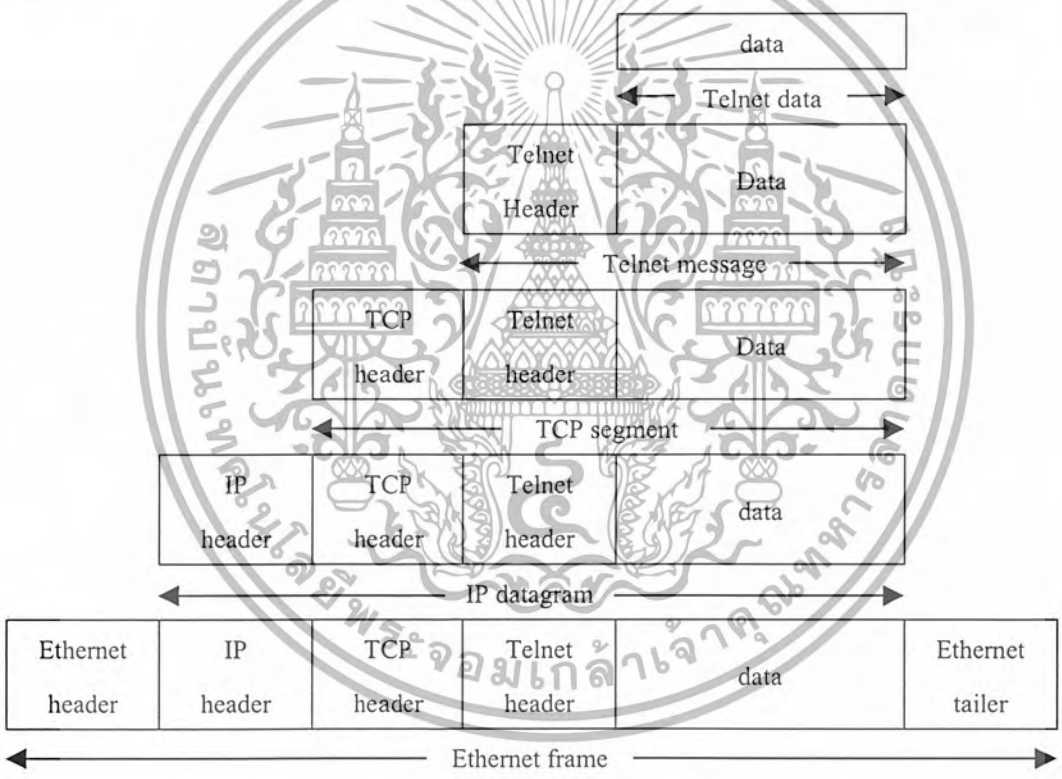
รูป 2-05 Secure Hash Algorithm

บทที่ 3
ภาพรวมของโปรแกรมเทลเน็ต

เทลเน็ต คือ โปรแกรมประยุกต์ที่ทำให้ผู้ใช้สามารถเข้าถึงระบบผ่านเทอร์มินอลใด ๆ เพื่อขอใช้บริการจากโฮสต์ใด ๆ ที่เปิดให้บริการ ผ่านทางเครือข่ายทีซีพี/ไอพี ในการติดต่อระหว่างเทอร์มินอลกับเครื่องให้บริการนั้น ๆ นั่นเอง

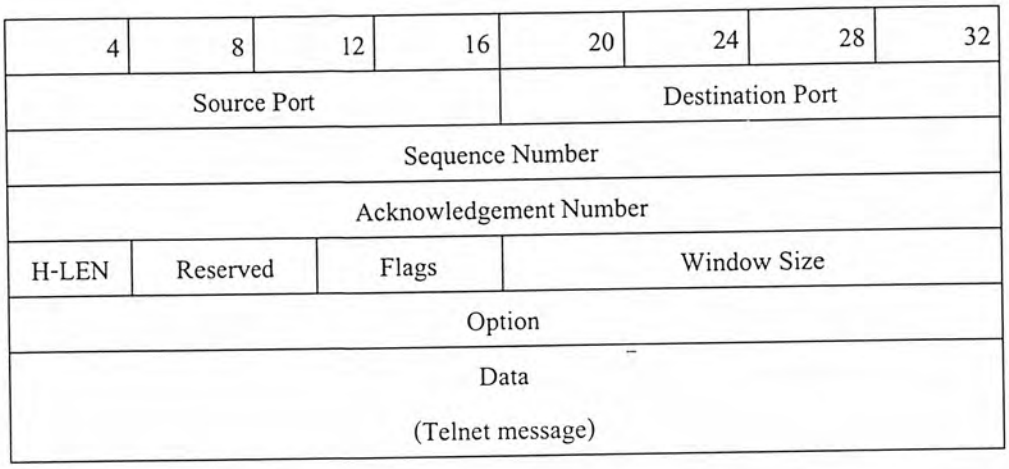
3.1 เทลเน็ตทำงานอย่างไร

เทลเน็ต เป็นโปรแกรมประยุกต์ที่ทำงานอยู่บนในชั้นแอปพลิเคชั่นของโพรโตคอลทีซีพี/ไอพี ดังนั้นข้อมูลจากผู้ใช้ก่อนที่จะส่งออกไปในระบบเครือข่ายต้องมีการเพิ่มส่วนหัวในชั้นต่าง ๆ ดังรูป 3-01

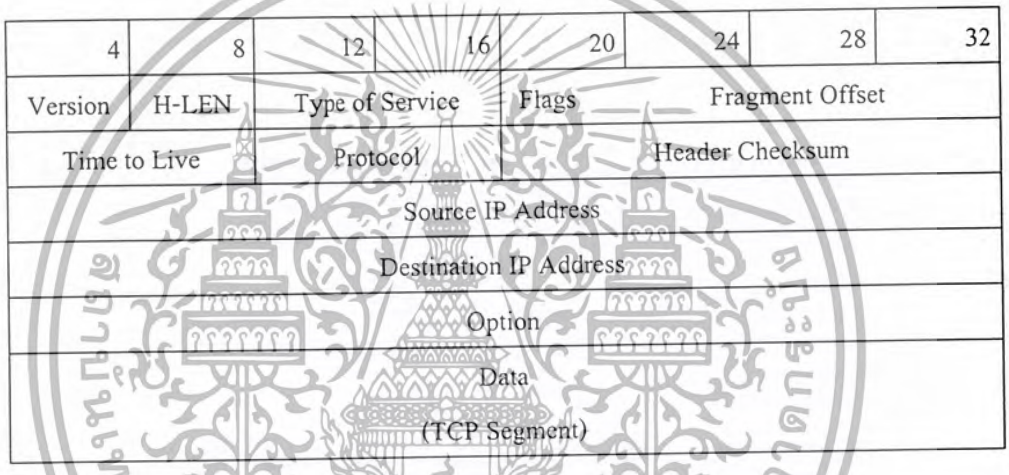


รูป 3-01 รูปแบบของข้อมูลเทลเน็ตที่ส่งในเครือข่าย

ส่วนทีซีพีพอร์ตบริการของเทลเน็ตนั้น ค่าที่กำหนดจาก IANA จะเป็นหมายเลข 23 ดังนั้นเมื่อมีการเรียกใช้โปรแกรมเทลเน็ต การเชื่อมต่อจะกระทำกับพอร์ต 23 ที่เครื่องเซิร์ฟเวอร์เสมอ



รูป 3-02 โครงสร้างของทีซีพีเซ็กเมนต์



รูป 3-03 โครงสร้างของไอพีดาต้าแกรม

3.2 เทลเน็ตมีหลักการทำงานเบื้องต้นอย่างไร

หลังจากที่ผู้ใช้พิมพ์คำสั่ง "telnet <host> จะมีขั้นตอนการทำงานดังนี้

1. ที่เครื่องของผู้ใช้หรือไคลเอนท์ เริ่มโปรแกรมเทลเน็ต

2. เทลเน็ตจากไคลเอนท์จะติดต่อกับเครื่องเซิร์ฟเวอร์ ถ้าติดต่อได้เซิร์ฟเวอร์จะตอบกลับ (ACK) แต่ถ้าไม่ได้ก็จะแจ้งว่าผิดพลาด (Abort) แล้วออกจากโปรแกรมไป

3. ถ้าติดต่อได้แล้ว ไคลเอนท์ก็จะขอทำการติดต่อกับเซิร์ฟเวอร์ โดยถ้าเซิร์ฟเวอร์พร้อมให้บริการแล้วก็จะส่งสัญญาณ (ACK) ตอบกลับมา พร้อมทั้งส่ง Escape Character ที่ใช้ในการทำเทอร์มินอลเสมือน (virtual terminal) มาด้วย

4. ไคลเอนท์เมื่อได้รับสัญญาณ (ACK) ตอบกลับแล้วจะรอข้อมูลจากผู้ใช้ป้อนชื่อและรหัสผ่าน แล้วจะทำการส่งข้อมูลเหล่านั้นไปให้กับเซิร์ฟเวอร์

5. เซิร์ฟเวอร์จะตรวจสอบชื่อและรหัสผ่าน และตอบกลับมาว่าพิสูจน์ตนผ่านหรือไม่ ถ้าผ่านก็จะเข้าสู่กระบวนการของเชลต่อไป

3.3 Network Virtual Terminal (NVT)

เนื่องจากเทลเน็ตในทางปฏิบัติ เมื่อผู้ใช้ล็อกอิน (login) กับเซิร์ฟเวอร์ใด ๆ ก็ตาม เทลเน็ตจะล็อกอินผ่าน NVT เพื่อให้ผู้ใช้สามารถใช้เทอร์มินอลของตนในการล็อกอินเข้าไปใช้บริการในเซิร์ฟเวอร์

3.3.1 โครงสร้างหลักและหลักการของ NVT

NVT เป็นหลักการที่ใช้ส่งข้อมูลระหว่างแอพพลิเคชันของเทอร์มินอล (terminal application) ที่อยู่ต่างเครื่องและต่างสิ่งแวดล้อมกัน ซึ่งได้แก่ ความแตกต่างในเรื่องของระบบปฏิบัติการ ตลอดจนถึงความแตกต่างในเรื่องโพรโตคอลที่ใช้ในระดับบน (upper-level) ของแอพพลิเคชันที่ทำงานอยู่ต่างเครื่องกันนั้น ซึ่งหลักการของ NVT นี้ ทำให้แอพพลิเคชันที่ทำงานอยู่บนไคลเอนท์และแอพพลิเคชันที่ทำงานอยู่บนเซิร์ฟเวอร์ ซึ่งมีสิ่งแวดล้อมที่ต่างกันสามารถติดต่อกันได้ เปรียบเสมือนโปรแกรมเทลเน็ตที่ทำงานอยู่บนไคลเอนท์ต้องติดต่อกับ telnetd ที่ทำงานอยู่บนเซิร์ฟเวอร์นั่นเอง

หลักการของการทำ NVT มีอยู่ว่า เรามองเทอร์มินอลที่ใช้งานแอพพลิเคชันทั้ง 2 ที่ต้องการติดต่อกันนั้นเป็น 2 เทอร์มินอล คือ Physical Terminal กับ Logical Terminal

Physical Terminal คือ เทอร์มินอลที่เรียกใช้แอพพลิเคชันนั้นจริง ๆ ซึ่ง Physical Terminal ที่เรียกใช้แอพพลิเคชันทั้ง 2 อาจแตกต่างกันในหลาย ๆ เรื่องดังที่ได้กล่าวมาแล้ว

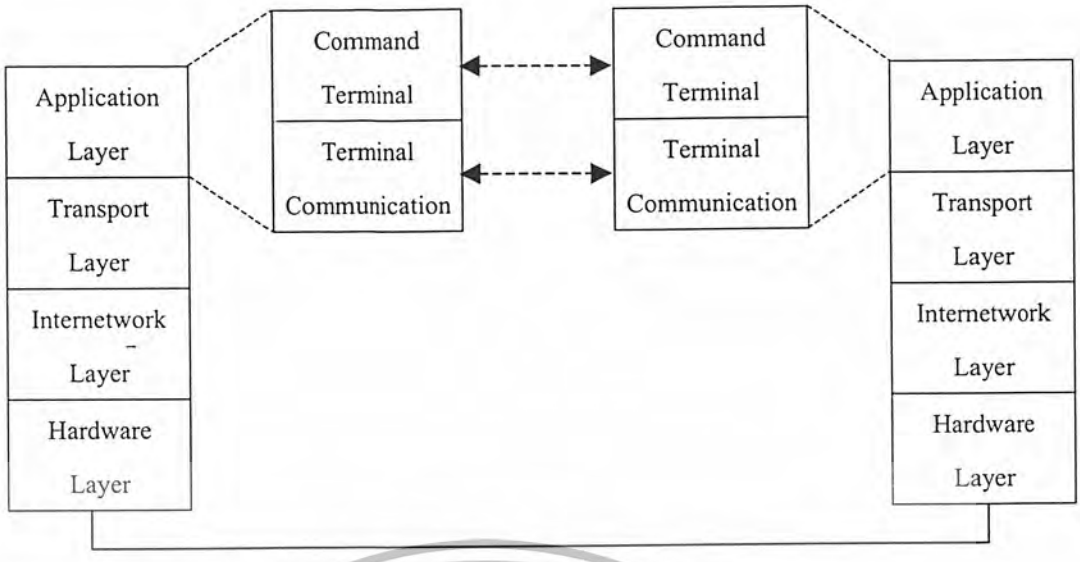
Logical Terminal คือ เทอร์มินอลในความคิด โดยเรากำหนดมาตรฐานต่าง ๆ ในการติดต่อให้เหมือนกันทั้ง 2 เครื่อง และเมื่อ Logical Terminal บนเครื่องทั้ง 2 เหมือนกัน จึงทำให้ Logical Terminal ทั้ง 2 เครื่องคุยกันได้ และ Logical Terminal ของแต่ละเครื่องก็จะเทียบ (map) ข้อมูลที่ติดต่อกันนั้น ให้อยู่ในรูปแบบที่ Physical Terminal ของเครื่องตนเองเข้าใจ ดังนั้นแอพพลิเคชันที่ทำงานอยู่บน Physical Terminal ที่ต่างกันจึงสามารถติดต่อและแลกเปลี่ยนข้อมูลกันได้ โดยไม่มีปัญหาเรื่องสภาพแวดล้อมของแต่ละเครื่องเข้ามาเกี่ยวข้องเลย ด้วยหลักการที่กล่าวทั้งหมดนี้ในทางปฏิบัติแล้วเราจะแบ่งชั้นแอพพลิเคชันออกเป็น 2 ชั้นย่อย (sublayer) ดังรูป 3-04

Command Terminal เป็นชั้นย่อย (sublayer) ที่เรียกใช้แอพพลิเคชันอยู่จริง โดยที่ชั้นย่อยนี้มีสภาพแวดล้อมต่าง ๆ ตามเครื่องที่ทำงานอยู่ ซึ่งชั้นย่อยนี้ ก็เปรียบเสมือน Physical Terminal นั่นเอง

Terminal Communication เป็นชั้นย่อยที่ทำหน้าที่แปลงสิ่งแวดล้อมที่ไม่เหมือนกันทั้ง 2 เครื่องให้เป็นมาตรฐานเดียวกัน เพื่อให้แอพพลิเคชันทั้ง 2 สามารถคุยกันได้และแปลงข้อมูลที่ติดต่อกันให้อยู่ในรูปแบบที่ Physical Terminal ของตนเองเข้าใจ ซึ่งชั้นย่อยนี้ก็เปรียบเสมือน Logical Terminal นั่นเอง

3.3.2 การทำงานและออปชันของ NVT ในเทลเน็ต

การทำงานเริ่มเมื่อผู้ใช้เรียกเทลเน็ตที่เทอร์มินอลของตนเอง ที่สามารถติดต่อเข้าไปใน NVT ของเซิร์ฟเวอร์ที่ต้องการติดต่อด้วยได้อย่างที่ได้กล่าวมาแล้ว NVT ที่ติดต่อด้วยนั้นทำให้ผู้ใช้เหมือนได้ใช้เทอร์มินอลบนเซิร์ฟเวอร์นั้นจริง ๆ แต่อุปกรณ์ที่ผู้ใช้เห็นบนเซิร์ฟเวอร์นั้นจริง ๆ แล้วเป็นเพียงการจำลองมาเท่านั้น



รูป 3-04 โครงสร้างการทำ NVT

เพื่อให้เห็นภาพได้ชัดเจนยิ่งขึ้น ลองนึกถึงอุปกรณ์ในส่วนของเทอร์มินอล (terminal device) จริง ๆ ดูว่าอะไรเป็นอุปกรณ์ที่เป็นอินพุต (เช่น แป้นพิมพ์) และอะไรเป็นอุปกรณ์ที่เป็นเอาต์พุต (เช่น จอภาพ เครื่องพิมพ์) ดังนั้น NVT จะจำลองอุปกรณ์ในส่วนของเทอร์มินอลจริง ๆ เหล่านั้น ซึ่งก็มีแป้นพิมพ์เป็นอินพุตและมีเครื่องพิมพ์เป็นเอาต์พุต บางข้อความที่เซิร์ฟเวอร์ต้องแสดงผลเครื่องพิมพ์จะพิมพ์ข้อความที่ได้รับมาจากเซิร์ฟเวอร์ และแป้นพิมพ์ก็จะสร้างข้อความที่ต้องส่งออกไปยังเครื่องเซิร์ฟเวอร์ เสมือนว่าเทอร์มินอลที่ใช้อยู่นั้น เป็นเทอร์มินอลจริง ๆ บนเครื่องเซิร์ฟเวอร์นั่นเอง

เนื่องจากในแต่ละเซิร์ฟเวอร์มักมีผู้ใช้จำนวนมาก ดังนั้นการทำให้ NVT ตอบสนองกับสภาพแวดล้อมของเทอร์มินอลของแต่ละผู้ใช้จึงทำได้ยาก วิธีแก้ไขคือ ใช้ออปชันต่าง ๆ ตามข้อตกลงที่แต่ละเทอร์มินอลและผู้ใช้แต่ละคนต้องการ โดยทำการตกลงกันในตอนที่ติดต่อกัน ซึ่งออปชันที่ NVT เตรียมไว้มีตัวอย่างดังนี้

- เปลี่ยนแบบตัวอักษรของ NVT
- เลือกได้ว่าให้ NVT ส่งตัวอักษรกลับมาให้ผู้ใช้เห็นบนจอภาพหรือไม่
- เลือกโหมดส่งข้อมูลเป็นทีละบรรทัดหรือทีละตัวอักษร

และยังมีออปชันอื่น ๆ อีก การขอทำออปชันต่าง ๆ เหล่านี้ สามารถเลือกว่าจะใส่เพิ่มหรือถอนออกก็ได้ ซึ่งการตกลงเจรจาที่จะทำออปชันนั้น ทำได้โดยใช้ลำดับของข้อความ WILL, WON'T, DO และ DON'T นั่นเอง

"WILL X" เป็นข้อความที่ส่งจากด้านใดด้านหนึ่ง โดยด้านที่ส่งนั้นได้บอกอีกด้านหนึ่งที่ติดต่อด้วยว่า ตนเองจะทำออปชัน X ซึ่งถ้าอีกด้านหนึ่งยอมรับการทำออปชันนั้น ก็ตอบกลับมาด้วย "DO X" แต่ถ้าอีกด้านไม่ยอมรับการทำออปชันนั้น ก็ตอบกลับมาว่า "DON'T X"

"DO X" เป็นข้อความที่ส่งโดยด้านใดด้านหนึ่ง โดยที่มันต้องการให้อีกด้านหนึ่งที่ทำการติดต่อด้วยทำออปชัน X ให้มัน ถ้าอีกด้านหนึ่งนั้นตอบรับการทำออปชัน X ก็ตอบกลับมาด้วย "WILL X" แต่ถ้าอีกด้านไม่ยอมทำออปชัน X ก็ตอบกลับไปว่า "WON'T X" นั่นเอง

3.4 โหมดการส่งข้อมูลของเทลเน็ต

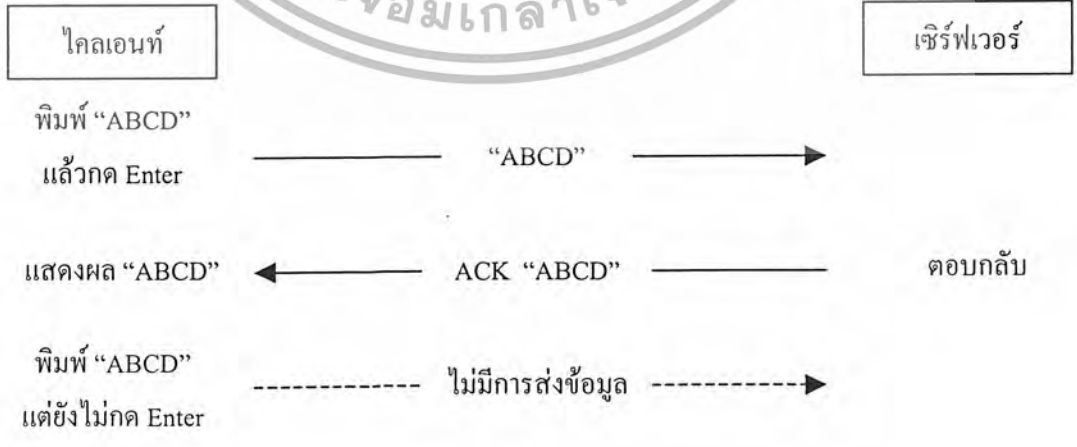
หลังจากที่ติดต่อระหว่างคำสั่งของเทลเน็ต (Telnet Command) บนไคลเอนท์กับกระบวนการของ Telnetd บนเซิร์ฟเวอร์แล้ว ก็มีการตกลงออปชันกัน หลังจากนั้นก็ทำการส่งข้อมูล ซึ่งโหมดการส่งข้อมูลที่เป็นคำสั่งของเทลเน็ตไปยังกระบวนการของ Telnetd มีอยู่ 2 โหมดคือ โหมดการส่งข้อมูลทีละบรรทัด (line-at-a-time) และโหมดการส่งข้อมูลทีละตัวอักษร (character-at-a-time)

3.4.1 โหมดการส่งข้อมูลทีละบรรทัด

การส่งข้อมูลในโหมดนี้ จะส่งข้อมูลทีละบรรทัด ส่งเมื่อผู้ใช้มีการกดคีย์ขึ้นบรรทัดใหม่ (new line) หลังจากส่งข้อมูลจากไคลเอนท์ไปให้เซิร์ฟเวอร์แล้ว เซิร์ฟเวอร์ก็จะตอบรับ (ACK) ข้อมูลที่ส่งมานั้นเท่านั้น จะไม่มีการส่งข้อความกลับมา (echo) เพื่อแสดงผลบนหน้าจอไคลเอนท์แต่อย่างใด เพราะการแสดงผลบนไคลเอนท์จะควบคุมด้วยตัวของไคลเอนท์เอง เนื่องจากไม่สามารถรอการส่งข้อความกลับมาจากเซิร์ฟเวอร์ได้ เพราะไคลเอนท์เองต้องรอการกดคีย์เพื่อขึ้นบรรรทัดใหม่ จึงจะส่งข้อมูล ดังนั้นถ้ารอข้อความที่ส่งกลับมาจากเซิร์ฟเวอร์แล้ว ผู้ใช้จะไม่เห็นตัวอักษรที่ตนพิมพ์เข้าไปจนกว่าผู้ใช้กดคีย์ขึ้นบรรทัดใหม่ ซึ่งถ้าเป็นเช่นนี้แล้วจะเป็นการไม่สะดวกที่ผู้ใช้แก้ข้อความที่พิมพ์ผิด เพราะผู้ใช้ยังไม่เห็นข้อความที่พิมพ์เข้าไป

ดังนั้นเพื่อไม่ให้เกิดเหตุการณ์ดังกล่าว การส่งข้อมูลในโหมดนี้จึงให้ไคลเอนท์เป็นเครื่องที่ควบคุมการแสดงผลเอง ดังนั้นการส่งข้อมูลในโหมดนี้จึงไม่มีปัญหาในการใช้คีย์พิเศษอย่าง Backspace หรือ Delete เพราะอย่างที่ได้กล่าวไปแล้วว่าการจัดการแสดงผลทำโดยเทอร์มินอลของผู้ใช้ และผู้ใช้ยังเป็นผู้กำหนดว่าส่งข้อมูลไปให้เซิร์ฟเวอร์เมื่อไหร่เองอีกด้วย

แต่การส่งข้อมูลทีละบรรทัดก็มีปัญหาคือ การส่งข้อมูลในโหมดนี้ เมื่อส่งข้อมูลไปแล้วเราไม่สามารถกลับไปแก้ไขข้อมูลเดิมได้อีก จึงทำให้ไม่สามารถใช้โปรแกรมประเภท word processor หรือ text editor ได้ในเทลเน็ตโหมดนี้ เพราะการส่งข้อมูลแบบนี้ไม่สามารถแก้ไขข้อมูลที่ส่งไปแล้วได้นั่นเอง ดังรูป 3-05



รูป 3-05 โหมดการส่งข้อมูลทีละบรรทัด

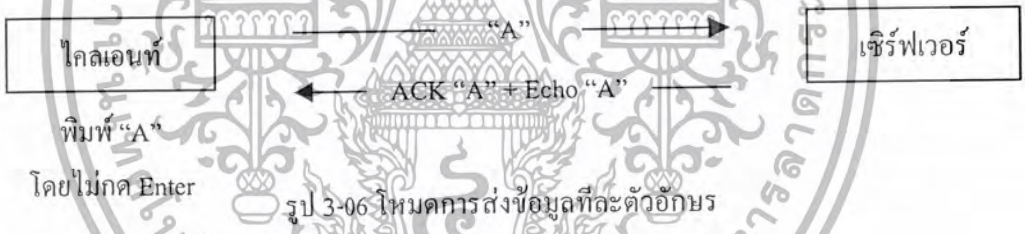
3.4.2 โหมดการส่งข้อมูลแบบทีละตัวอักษร

การส่งข้อมูลในโหมดนี้จะส่งข้อมูลทีละตัวอักษร คือส่งทันทีที่ผู้ใช้มีการกดคีย์ใด ๆ และส่งข้อมูลทีละตัวอักษร ซึ่งการส่งข้อมูลในโหมดนี้คำสั่งของเทลเน็ตที่ทำงานอยู่บนไคลเอนท์จะยังไม่แสดงผลข้อมูลที่ส่งไปยังเซิร์ฟเวอร์บนจอภาพทันที แต่จะรอให้เซิร์ฟเวอร์ตอบรับและส่งข้อความกลับมาให้จึงแสดงผลบนจอภาพ

ข้อดีของการส่งข้อมูลในโหมดนี้ คือผู้ใช้สามารถแก้ไขข้อมูลที่ส่งไปแล้วได้ง่ายกว่าการส่งข้อมูลในโหมดการส่งแบบทีละบรรทัด เพราะข้อมูลที่ส่งไปแล้วนั้นมีขนาดเพียง 1 ตัวอักษร ดังนั้นเราจึงส่งข้อมูลไปแก้ไขข้อมูลที่ส่งไปแล้วเหล่านั้นได้ แล้วด้วยเหตุนี้จึงทำให้การส่งข้อมูลในโหมดนี้สามารถใช้งานโปรแกรมพวก word processor หรือ text editor ได้ จึงทำให้การส่งข้อมูลในโหมดนี้เป็นที่นิยมใช้กันในปัจจุบัน

แต่การส่งข้อมูลในโหมดการส่งแบบทีละตัวอักษรก็มีปัญหาคือ การแสดงผลบนจอภาพของการส่งข้อมูลในโหมดนี้นั้นต้องแสดงจากข้อความที่ส่งกลับมากจากเซิร์ฟเวอร์ ดังนั้นการแสดงผลบางอย่างจึงไม่สามารถทำได้ เช่น เมื่อผู้ใช้กดคีย์ Backspace แล้วโฮสต์ส่งแอสกีของ Backspace กลับมาให้ไคลเอนท์ ซึ่งไม่ได้เป็นไปตามที่ผู้ใช้ต้องการ แต่สิ่งที่ผู้ใช้ต้องการคือให้เลื่อนเคอร์เซอร์ (cursor) กลับไปหนึ่งตัวอักษรและลบอักขระตัวนั้นด้วย

ดังนั้นจึงต้องมีการทำเทอร์มินอลเสมือนที่กล่าวถึงในหัวข้อต่อไป



รูป 3-06 โหมดการส่งข้อมูลทีละตัวอักษร

3.5 เทอร์มินอลเสมือน (Virtual Terminal)

เหตุผลในการทำเทอร์มินอลเสมือนก็เพราะการส่งข้อมูลในโหมดการส่งข้อมูลทีละตัวอักษร การแสดงผลบนจอภาพของไคลเอนท์จะควบคุมโดยเซิร์ฟเวอร์ แต่เนื่องจากไคลเอนท์และเซิร์ฟเวอร์อยู่ห่างไกลกัน การติดต่อก็ทำได้โดยผ่านเครือข่ายที่เชื่อมเครื่องทั้งสอง ซึ่งการส่งข้อมูลผ่านเครือข่ายนี้เอง ทำให้เซิร์ฟเวอร์ไม่สามารถควบคุมการแสดงผลของไคลเอนท์ด้วยการส่งข้อความกลับมาเพียงอย่างเดียว ดังนั้นจึงต้องมีการทำเทอร์มินอลเสมือน เช่น เมื่อผู้ใช้กดคีย์ Backspace แทนที่เซิร์ฟเวอร์ส่งตัวอักษรที่เป็น Backspace กลับมา ก็ส่งคำสั่งให้ไคลเอนท์ย้ายตำแหน่งเคอร์เซอร์ถอยหลังกลับ 1 ตัวอักษรแทน เป็นต้น ซึ่งคำสั่งที่เซิร์ฟเวอร์ใช้ทำเทอร์มินอลเสมือนเหล่านี้นั้น มีลักษณะเป็นชุดของคำสั่งต่าง ๆ โดยเริ่มต้นด้วย Escape chracter ซึ่งตกลงกันไว้ระหว่างเซิร์ฟเวอร์และไคลเอนท์ตั้งแต่ตอนที่ติดต่อกันในตอนแรก (ในขณะที่ตกลงออปชันต่าง ๆ นั่นเอง) ซึ่งชุดคำสั่งที่ใช้ในการทำเทอร์มินอลเสมือนเหล่านี้ ได้มีการกำหนด

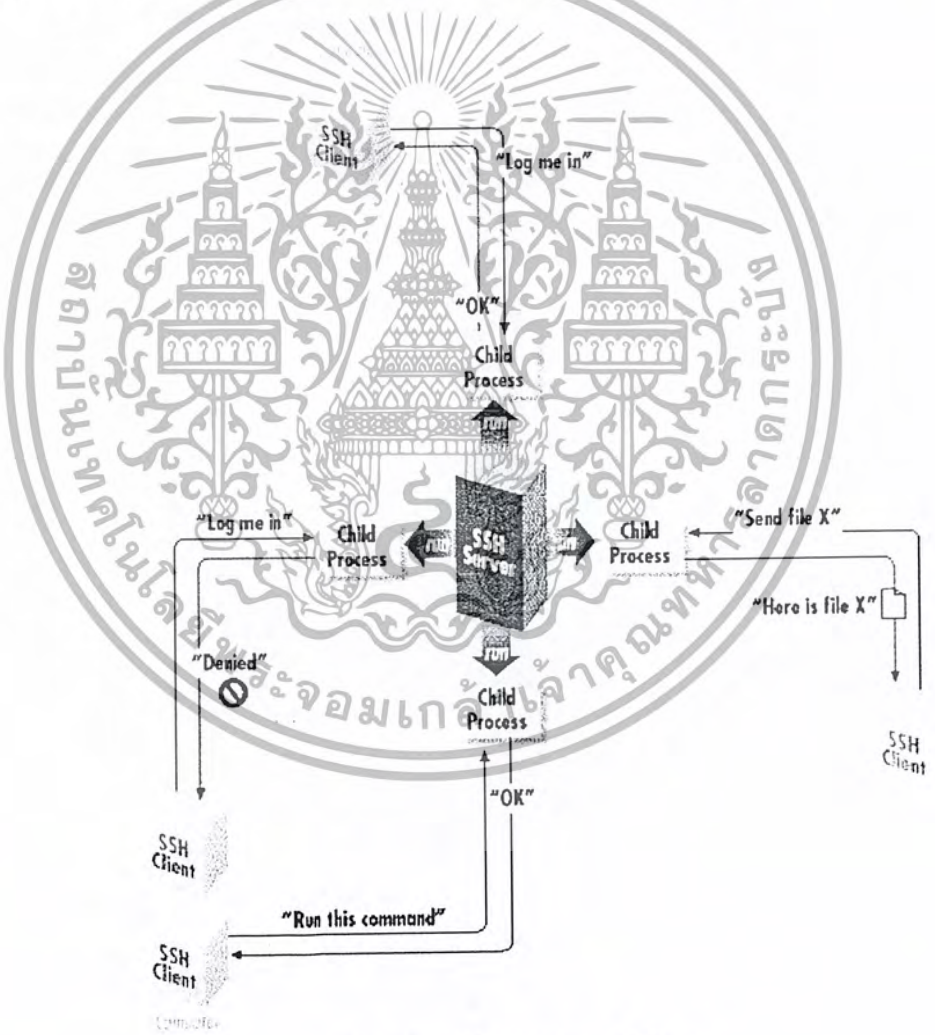
ไว้เป็นมาตรฐานต่าง ๆ ไว้มากมายหลายมาตรฐาน แต่ที่นิยมใช้กันมากที่สุด ได้แก่ มาตรฐานตระกูล VT ที่ชื่อว่า VT100

บทที่ 4
ซีเคียวเชล

4.1 ภาพรวมของเซ็กเคียวเชล

ซีเคียวเชล (Secure Shell : SSH) เป็นโพรโตคอลและโปรแกรมด้านความปลอดภัยของเครือข่ายที่ได้รับความนิยม และมีประโยชน์มาก เวลาส่งข้อมูลจากคอมพิวเตอร์เข้าไปในระบบเครือข่าย ซีเคียวเชลจะเข้ารหัสลับให้อัตโนมัติ และเมื่อข้อมูลถึงปลายทางก็จะถอดรหัสลับออกมาเอง ผลก็คือผู้ใช้ไม่จำเป็นต้องไปยุ่งเกี่ยวกับการเข้ารหัสลับเลย ก่อนหน้านี้ทำงานอย่างไรปัจจุบันก็เหมือนเดิม

ซีเคียวเชลใช้สถาปัตยกรรมแบบไคลเอนท์/เซิร์ฟเวอร์ ดังรูป 4-01 เซิร์ฟเวอร์ซีเคียวเชลจะรอรับคำร้องจากไคลเอนท์ ว่า "ขอเข้าสู่ระบบ", "ส่งไฟล์มา", หรือ "เรียกคำสั่งนี้" ซึ่งการสื่อสารทั้งหมดระหว่างเซิร์ฟเวอร์และไคลเอนท์จะเข้ารหัสลับอย่างปลอดภัย และปราศจากการแก้ไข



รูป 4-01 สถาปัตยกรรมของซีเคียวเชล

4.1.1 การเข้ารหัสลับ (Encryption)

การสื่อสารทางเครือข่ายคอมพิวเตอร์ปกติจะไม่มีการเข้ารหัสลับ ดังนั้นใครก็ตามที่อยู่ในระบบเครือข่าย สามารถดักฟังข้อมูลที่ผ่านไปมาได้ แม้ว่าเครือข่ายที่เป็นสวิตช์จะแก้ปัญหาได้บ้าง แต่ก็ยังคงมีปัญหานี้อยู่

ซีเคียวเชลจะเข้ารหัสลับข้อมูลที่จะส่งเข้าไปในเครือข่าย โดยจะเข้ารหัสลับระหว่างต้นทางและปลายทางด้วยการเข้ารหัสลับแบบใช้กุญแจเดี่ยว อัลกอริทึมที่ใช้เป็นได้ทั้งอัลกอริทึมที่เป็นมาตรฐานทั่วไป เช่น DES, 3DES และ AES เป็นต้น หรืออาจใช้เป็นอัลกอริทึมที่พัฒนาขึ้นมาเองก็ได้

4.1.2 ความถูกต้องของข้อมูล (Integrity)

ข้อมูลที่ส่งในเครือข่ายอาจเกิดการสูญหายได้ อาจจะเกิดจากสัญญาณรบกวน หรือข้อมูลมากเกินกว่าที่เครือข่ายรับได้ และบางครั้งอาจมีผู้ไม่ประสงค์ดีใช้จุดนี้โจมตีได้ โดยใช้วิธีที่เรียกว่าการโจมตีด้วยวิธีเล่นซ้ำ (replay attack) โดยการบันทึกการสื่อสารก่อนหน้านี้ แล้วติดต่อเข้าไปใหม่ จากนั้นเล่นซ้ำการสื่อสารครั้งก่อนหน้า

สำหรับซีเคียวเชลจะใช้ CRC-32 ในเวอร์ชันที่ 1 หรือแฮชฟังก์ชันในเวอร์ชันที่ 2 เพื่อตรวจสอบถึงการเปลี่ยนแปลงของข้อมูล

4.1.3 การพิสูจน์ตน (Authentication)

การพิสูจน์ตนเป็นการตรวจสอบว่าบุคคลนั้นเป็นตัวจริงตามที่ได้กล่าวอ้าง ซึ่งหากเป็นการพิสูจน์ตนในชีวิตประจำวัน เช่น เทียบกับรูปในบัตรประชาชน เทียบลายนิ้วมือ หรืออาจเป็นตรวจจาก DNA

ในซีเคียวเชลจะมีการพิสูจน์ตน 2 ทาง คือ ไคลเอนท์พิสูจน์กับเซิร์ฟเวอร์ (server authentication) และเซิร์ฟเวอร์พิสูจน์กับไคลเอนท์ (user authentication) การพิสูจน์ตนกับเซิร์ฟเวอร์ป้องกันการปลอมตัวเป็นเซิร์ฟเวอร์ และยังป้องการการโจมตีจากบุคคลตรงกลาง (man-in-the-middle attack) ซึ่งแอบซ่อนอยู่ระหว่างไคลเอนท์กับเซิร์ฟเวอร์ โดยเมื่อติดต่อกับเซิร์ฟเวอร์จะปลอมเป็นไคลเอนท์ และเมื่อติดต่อกับไคลเอนท์จะปลอมเป็นเซิร์ฟเวอร์

สำหรับการพิสูจน์ของเซิร์ฟเวอร์ ในซีเคียวเชลแต่ละเซิร์ฟเวอร์จะมีโฮสต์คีย์ ซึ่งเป็นกุญแจของการเข้ารหัสลับแบบใช้กุญแจสาธารณะ เพื่อใช้ในการยืนยันตัวเซิร์ฟเวอร์

และการพิสูจน์ตนของไคลเอนท์มีหลายวิธี แต่วิธีที่ใช้กันทั่วไปคือใช้รหัสผ่าน ซีเคียวเชลจะเข้ารหัสลับให้รหัสผ่านก่อนที่จะส่งไปในเครือข่าย ซึ่งเป็นการปรับปรุงจากโพรโตคอลเดิมอย่างเทลเน็ต หรือเอฟทีพี ที่จะส่งรหัสผ่านเป็นตัวอักษรเปล่า นอกจากนี้อาจพิสูจน์ตนโดยใช้กุญแจสาธารณะ, รหัสผ่านแบบใช้ครั้งเดียว และ PAM ก็ได้ โดยอัลกอริทึมและวิธีการที่ใช้จะขึ้นอยู่กับการคอนฟิก

4.1.4 การบีบอัดข้อมูล (Compression)

มีการบีบอัดข้อมูลที่ส่งเข้าไปในเครือข่าย มีประโยชน์ 3 ประการคือ ข้อมูลลดลงเวลาที่ใช้ในการประมวลผลการเข้ารหัสลับจะน้อยลง อัลกอริทึมที่มีผลเห็นได้ชัดคือ 3DES และการบีบอัดข้อมูลทำให้

รูปแบบ (pattern) ของข้อมูลที่ซ้ำกันลดลง อัลกอริทึมเข้ารหัสลับบางอัลกอริทึม หากข้อมูลมีรูปแบบที่ซ้ำกัน จะแคร็ก (crack) ได้ง่ายขึ้น และอีกประการหนึ่งคือ ข้อมูลก่อนเข้ารหัสลับมีขนาดเล็ก เมื่อผ่านอัลกอริทึมส่วนมากแล้ว มักจะได้ข้อมูลที่เข้ารหัสลับมีขนาดเล็กตามไปด้วยเช่นกัน ซึ่งจะส่งเข้าไปในระบบเครือข่ายได้เร็วขึ้นด้วย

สำหรับการบีบอัดข้อมูลสนับสนุนทั้งในเวอร์ชันที่ 1 และ 2 โดยอัลกอริทึมบีบอัดข้อมูลที่ใช้เป็น zlib

4.2 ขั้นตอนการติดต่อ

4.2.1 การตรวจสอบเวอร์ชัน

หลังจากที่ไคลเอนท์ร้องขอการเชื่อมต่อและทำ 3-way handshake แล้วเซิร์ฟเวอร์จะส่งข้อมูลเวอร์ชันของเซิร์ฟเวอร์มาให้ไคลเอนท์เช่น

SSH-1.99-OpenSSH_3.7.1p1\n

SSH-protocolversion-softwareversion\n

โดยจะแบ่งเป็น 3 ส่วนคือส่วนแรกบอกว่าเป็น SSH คือซีเคียวเชล ส่วนที่สองจะบอกเวอร์ชันที่สนับสนุน และส่วนสุดท้ายจะบอกชื่อและเวอร์ชันของโปรแกรม ในแต่ละส่วนจะถูกคั่นด้วยเครื่องหมายขีด (-) และจบด้วยอักขระขึ้นบรรทัดใหม่ซึ่งตรงกับแอสกี 10 หรืออาจเป็นแอสกี 13 กับแอสกี 10 ก็ได้

สำหรับเวอร์ชันของโพรโตคอลจะมี 2 เวอร์ชันคือ 1.x กับ 2.x สำหรับในตัวอย่างนี้ คือ 1.99 ซึ่งสนับสนุนทั้ง ในเวอร์ชันที่ 1 และ 2 เหตุผลที่เป็น 1.99 เพื่อที่จะให้ไคลเอนท์เก่ารู้ว่าสนับสนุนเวอร์ชันที่ 1 และในไคลเอนท์ที่สนับสนุนเวอร์ชันที่ 2 จะได้รู้ว่า 1.99 ก็คือ 2 นั่นเอง

และไคลเอนท์ก็จะตอบกลับในลักษณะเดียวกันเช่น

SSH-2.0-PuTTY-Snapshot-2003-12-04\n

ซึ่งจากข้อมูลของไคลเอนท์จะทราบได้ว่าไคลเอนท์จะใช้โพรโตคอลเวอร์ชัน 2 ในการเชื่อมต่อ แต่ถ้าหากไคลเอนท์ตอบมาเป็น

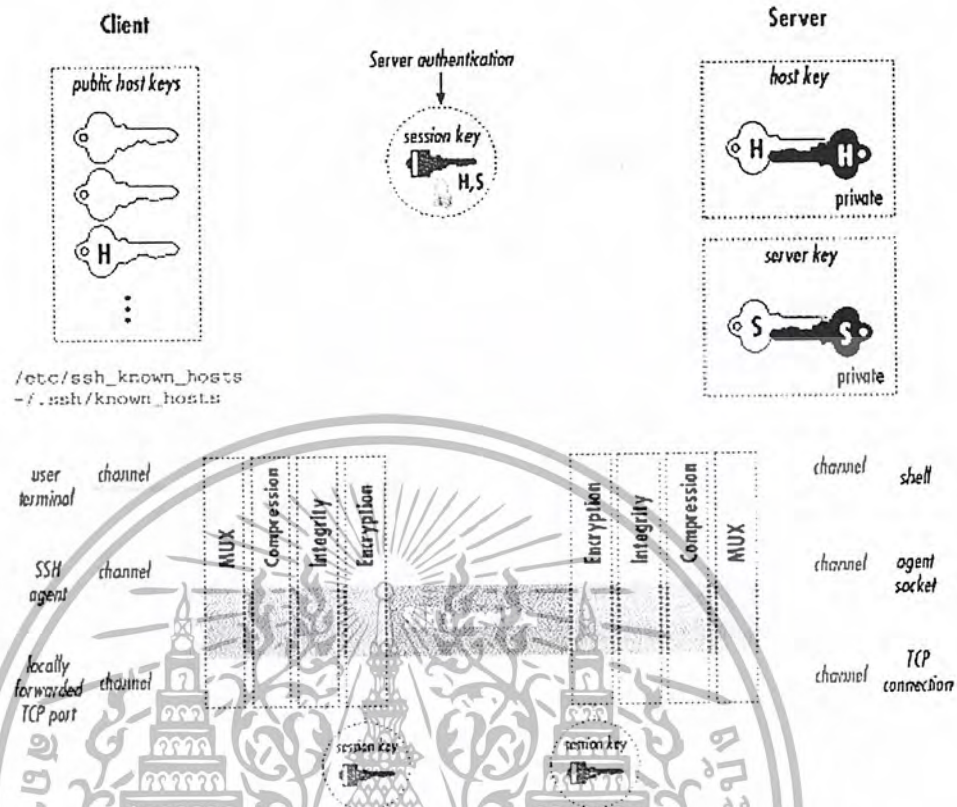
SSH-1.5-PuTTY-Snapshot-2003-12-04\n

หมายความว่าไคลเอนท์ใช้โพรโตคอลเวอรชัน 1 ในการเชื่อมต่อ นั่นก็คือเซิร์ฟเวอร์จะเลือกได้ก่อนว่าจะใช้เวอร์ชันไหน หากเซิร์ฟเวอร์ไม่เลือกคือเป็น 1.99 ไคลเอนท์จะเป็นตัวเลือกเวอร์ชัน

การทำงานหลังจากนี้ในแต่ละเวอร์ชันจะต่างกัน ซึ่งโครงงานนี้จะไม่ลงในรายละเอียดของซีเคียวเชลเวอร์ชันที่ 1 มากนัก แต่จะเน้นในเวอร์ชันที่ 2 ซึ่งเป็นเวอร์ชันที่โครงงานนี้สนับสนุน

4.2.2 ซีเคียวเชลเวอร์ชันที่ 1

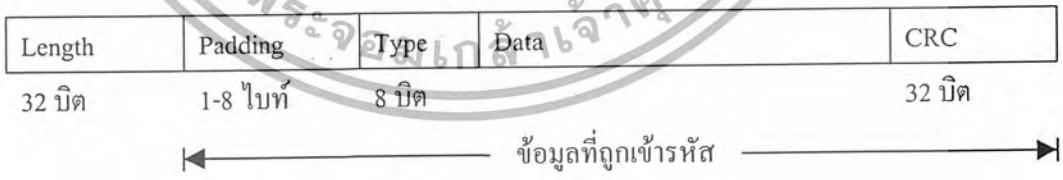
การทำงานของซีเคียวเชลเวอร์ชันที่ 1 แสดงอยู่ในรูป 4-2



รูป 4-02 สถาปัตยกรรมของซีเคียวเชลเวอร์ชันที่ 1

4.2.2.1 เซิร์ฟเวอร์และไคลเอนท์เปลี่ยนมาใช้ Binary Packet Protocol

หลังจากที่มีการแลกเปลี่ยนเวอร์ชันและรู้ว่าเป็นเวอร์ชันที่ 1 ทั้งเครื่องให้บริการและเครื่องผู้ใช้บริการจะส่งแพ็กเก็จที่มีรูปแบบเฉพาะ ซึ่งเรียกว่า Binary Packet Protocol โดยมีรูปแบบดังรูป 4-03



รูป 4-03 Binary Packet Protocol

- Length เป็นความยาวของแพ็กเก็ต มีขนาด 32 บิต ไม่นับรวมฟิลด์ความยาวและ Padding
- Padding เป็นข้อมูลที่สุ่มขึ้น ขนาด 1-8 ไบต์ (มีค่าเป็น 0 ถ้าไม่มีการเข้ารหัสข้อมูล) จำนวนของ padding มีขนาด (8 - (Length mod 8)) ไบต์
- Type มีขนาด 8 บิต
- Data เป็นข้อมูลไบนารีที่ส่งมีขนาดเท่ากับ Length – 5
- CRC ใช้ในการตรวจสอบความถูกต้องของข้อมูล

4.2.2.2 เซิร์ฟเวอร์ส่งกุญแจและอัลกอริทึมที่สนับสนุน

ถัดมาเซิร์ฟเวอร์จะส่งข้อมูลที่ใช้ในการพิสูจน์ตนและเข้ารหัสลับให้กับไคลเอนท์ ซึ่งประกอบไปด้วย

- กุญแจโฮสต์ เพื่อในการพิสูจน์ตนของเซิร์ฟเวอร์
- กุญแจเซิร์ฟเวอร์ เพื่อใช้สร้างการเชื่อมต่อที่ปลอดภัย
- ข้อมูลที่สุ่มขึ้นมา 8 ไบท์ เรียกว่า check bytes เพื่อใช้ป้องกันการโจมตีด้วยการปลอมไอพี โดยไคล์เอนท์ต้องส่งข้อมูลนี้กลับมาด้วย
- รายการของอัลกอริทึมเข้ารหัสลับ การบีบอัดข้อมูล และการพิสูจน์ตนที่เซิร์ฟเวอร์สนับสนุน

ในขั้นตอนนี้ไคลเอนท์จะเอากุญแจโฮสต์มาเทียบกับรายชื่อและกุญแจโฮสต์ที่เคยติดต่อว่าตรงกับของเดิมหรือไม่ ถ้าไม่ตรงหรือไม่มีก็จะถามผู้ใช้ว่าจะเชื่อถือกุญแจใหม่หรือไม่ ถ้าไม่เชื่อถือก็จะตัดการเชื่อมต่อ ตอนนี้ถือว่าเชื่อถือการเชื่อมต่อดำเนินต่อไป

4.2.2.3 ไคลเอนท์ส่งกุญแจเซสชัน

ไคลเอนท์จะสุ่มกุญแจขึ้นมาเพื่อใช้ในการเข้ารหัสลับแบบกุญแจเดี่ยวกับเซสชัน โดยจะเรียกกุญแจนี้ว่ากุญแจเซสชัน (session key)

การส่งกุญแจเซสชันไปให้เซิร์ฟเวอร์ จะต้องผ่านการเข้ารหัสลับสองชั้นคือ เข้ารหัสลับด้วยกุญแจโฮสต์ของเซิร์ฟเวอร์ และกุญแจเซิร์ฟเวอร์ แล้วจึงส่งกุญแจเซสชันที่เข้ารหัสแล้วไปพร้อมกับ check bytes และอัลกอริทึมที่เลือกใช้

4.2.2.4 เริ่มต้นการเข้ารหัสลับ

เซิร์ฟเวอร์จะถอดรหัสลับเพื่อนำกุญแจเซสชันมาเข้ารหัสลับเซสชัน ผู้ที่ไม่ใช่เซิร์ฟเวอร์ก็จะไม่สามารถถอดรหัสได้ และการเชื่อมต่อที่ปลอดภัยก็ได้เริ่มขึ้น

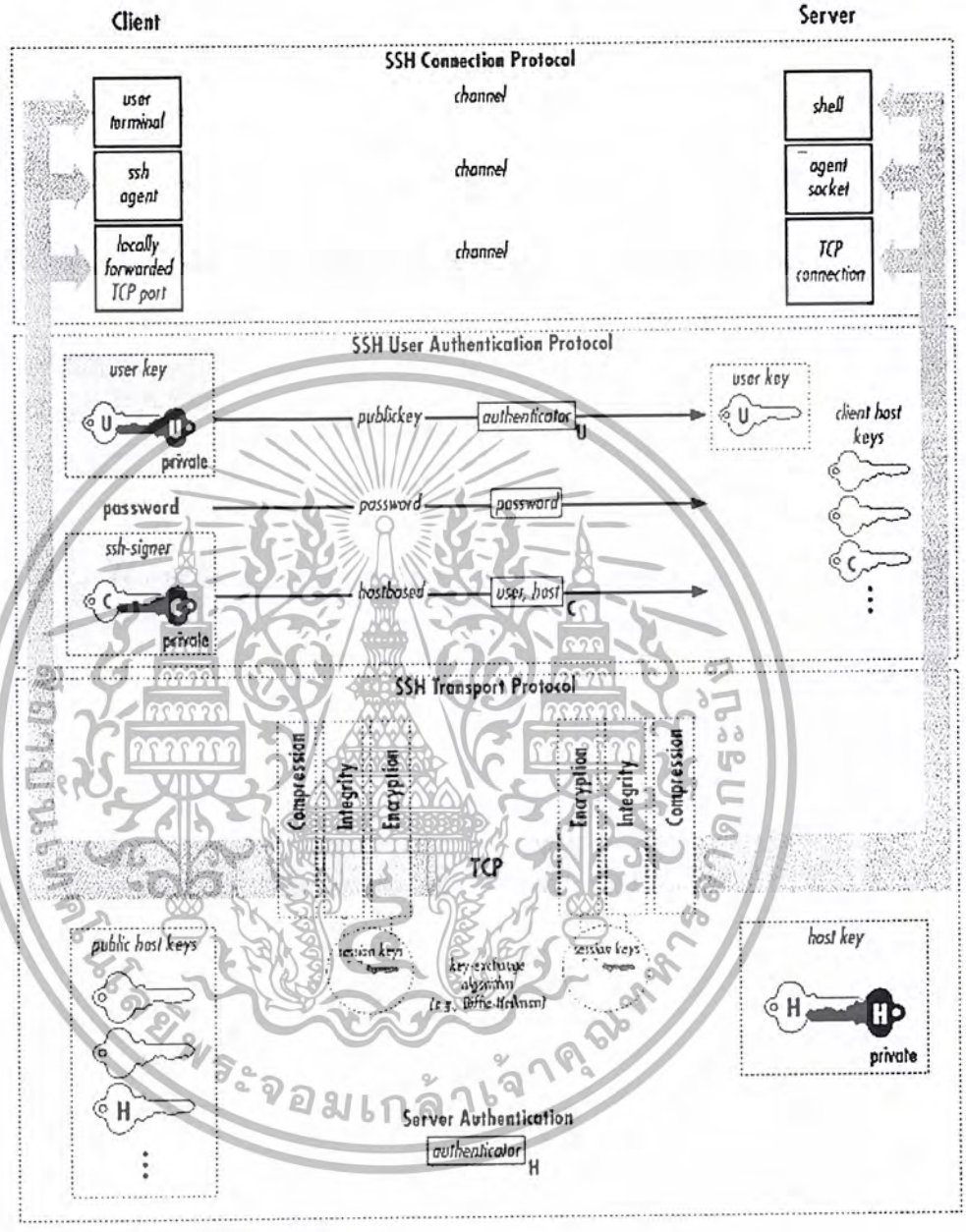
4.2.2.5 การพิสูจน์ตนของไคลเอนท์

ในซีเคียวเชลเวอร์ชันที่ 1 ไคลเอนท์จะพยายามพิสูจน์ตนแต่ละวิธีไปเรื่อยจนกว่าเจอวิธีที่สำเร็จหรือล้มเหลวทั้งหมด โดยในเวอร์ชัน SSH-1.5 จะเรียงลำดับดังนี้

1. Kerberos
2. Rhosts
3. RhostsRSA
4. Public-key
5. TIS
6. Password

4.2.3 ซีเคียวเชลเวอร์ชันที่ 2

ในเวอร์ชันที่ 2 นี้ ซีเคียวเชลประกอบด้วย 4 มาตรฐานหลัก โดยในปัจจุบันยังคงเป็นมาตรฐานร่าง (draft standard) เป็นโครงสร้างดังรูป 4-04 และรูป 4-05



รูป 4-04 สถาปัตยกรรมของซีเคียวเชลเวอร์ชันที่ 2

- **SSH Protocol Architecture (SSH-ARCH)** เป็นตัวหลักที่ครอบคลุมโพรโตคอลตัวอื่นๆ
- **SSH Transport Layer Protocol (SSH-TRANS)** กำหนดความรับผิดชอบในระดับของทรานสปอร์ต ซึ่งทำงานอยู่บนทีซีพี/ไอพี รับผิดชอบการเข้ารหัสลับ, การพิสูจน์ตนของเซิร์ฟเวอร์ และการตรวจสอบความถูกต้องของข้อมูล

● **SSH Authentication Protocol (SSH-AUTH)** จะรับผิดชอบการพิสูจนตนของผู้ใช้หรือไคลเอนท์ด้วยกุญแจสาธารณะ, รหัสผ่าน, และวิธีโฮสต์เบส โพรโตคอลนี้จะทำงานอยู่บน SSH-TRANS และจะทำการพิสูจน์ตนเพียงแค่ครั้งเดียวเพื่อใช้ใน SSH-CONN ต่อไป

● **SSH Connection Protocol (SSH-CONN)** จะกล่าวถึงการทำอินเทอร์แอกทีฟเซสชัน, การเรียกใช้งานคำสั่งจากระยะไกล, การทำทีซีพี/ไอพีพอร์ดฟอร์เวิร์ด (TCP/IP port forwarding) และการทำ X11 ฟอร์เวิร์ด (X11 forwarding) การสื่อสารทั้งหมดต้องทำมัลติเพล็กซ์เพื่อส่งในช่องทางเดียว โพรโตคอลนี้ออกแบบมาให้ทำงานอยู่บน SSH-TRANS และทำงานหลังจากผ่าน SSH-AUTH แล้ว

<table>
<tr><td colspan="2">แอพพลิเคชัน (เช่น ssh, sshd, scp, sftp,sftp_server)</td></tr>
<tr><td>SSH Authentication Protocol (SSH-AUTH)
การพิสูจน์ตนของไคลเอนท์
กุญแจสาธารณะ
รหัสผ่าน
โฮสต์เบส
การเปลี่ยนรหัสผ่าน</td><td>SSH Connection Protocol (SSH-CONN)
การทำทีซีพี/ไอพีพอร์ดฟอร์เวิร์ด
อินเทอร์แอกทีฟเซสชัน
การเรียกใช้งานคำสั่งจากระยะไกล
โฟล์คอนโทรล (flow control)
เทอร์มินอลเทียม (pseudo-terminal)
การจัดการเทอร์มินอล – ขนาดวินโดวส์
บีบอัดข้อมูล</td></tr>
<tr><td colspan="2">SSH Transport Layer Protocol (SSH-TRANS)
การพิสูจน์ตนของเซิร์ฟเวอร์
การเจรจาอัลกอริทึม
แลกเปลี่ยนกุญแจเซสชัน
การเข้ารหัสลับ
การตรวจสอบความถูกต้อง
เซสชันไอดี</td></tr>
<tr><td colspan="2">ทีซีพี</td></tr>
</table>

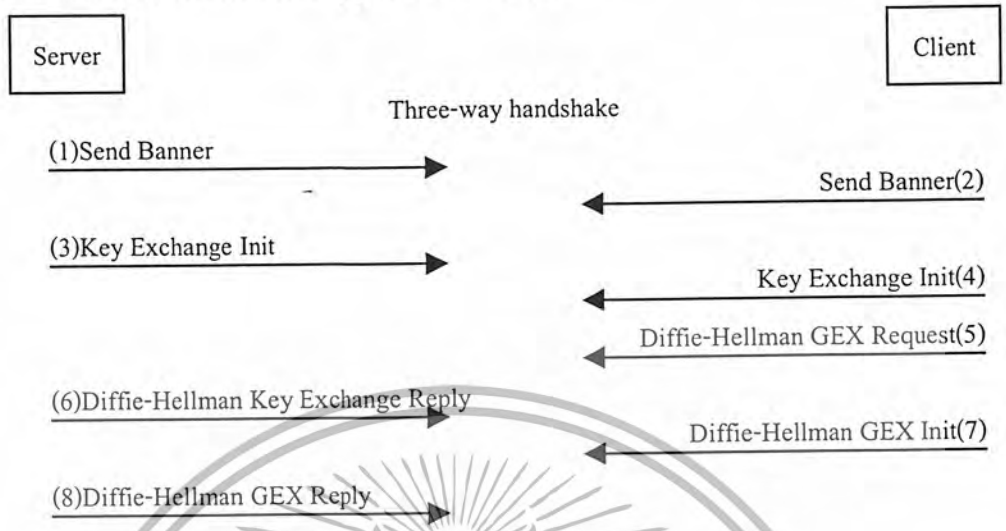
รูป 4-05 ความสัมพันธ์ของโพรโตคอลซีเคียวเชลเวอร์ชันที่ 2

ข้อแตกต่างของซีเคียวเชลเวอร์ชันที่ 1 และเวอร์ชันที่ 2 ได้แก่

- เพิ่มการเจรจาเรื่องอัลกอริทึมระหว่างเซิร์ฟเวอร์และไคลเอนท์
- สนับสนุนการแลกเปลี่ยนกุญแจหลายแบบ
- มีใบรับรองของกุญแจสาธารณะ
- การพิสูจน์ตนยืดหยุ่นมากขึ้น
- มีการตรวจสอบความถูกต้องที่ครอบคลุมมากขึ้น
- มีการเปลี่ยนกุญแจเซสชันเป็นระยะ

4.2.3.1 การสร้างการเชื่อมต่อที่ปลอดภัย

ทำได้โดยทั้งสองฝ่ายแลกเปลี่ยนลิสต์ของอัลกอริทึมที่สนับสนุน (3,4) จากนั้นใช้หลักการของ Diffe-Hellman (5-8) เพื่อให้ได้มาซึ่งกุญแจเซสชันที่ตรงกัน



รูป 4-06 แสดงการสร้างการเชื่อมต่อที่ปลอดภัย

4.2.3.2 การเรียกชื่ออัลกอริทึม

ในซีเคียวเชลเวอร์ชันที่ 2 จะใช้ชื่อเรียกอัลกอริทึมเป็นสตริงแทนที่จะเป็นไบนารีโคดเหมือนกับในเวอร์ชันที่ 1 ซึ่งจะยืดหยุ่นมากขึ้น ทำให้สามารถเพิ่มอัลกอริทึมได้ ทั้งที่เป็นอัลกอริทึมสาธารณะ และอัลกอริทึมที่พัฒนาเพื่อใช้กันภายในองค์กรเอง โดยการเรียกชื่ออัลกอริทึมนี้จะใช้ทั้งอัลกอริทึมเข้ารหัสลับ พิสูจน์ตน บีบอัดข้อมูล แลกเปลี่ยนกุญแจ และแฮชฟังก์ชัน

โดยอัลกอริทึมที่ไม่มีเครื่องหมายแอดไซน์ (@) จะถูกกำหนดโดย IANA เช่น 3des-cbc, sha-1, hmac-sha1 สำหรับอัลกอริทึมที่พัฒนาขึ้นเองให้ใช้เป็น name@domainname เช่น cipher-x@ce.kmitl.ac.th โดยชื่อนั้นต้องเป็นตัวอักษรแอสกี ยกเว้นแอดไซน์และคอมมา

4.2.3.3 การแลกเปลี่ยนกุญแจเซสชันและกุญแจเซิร์ฟเวอร์

ซีเคียวเชลเวอร์ชันที่ 1 ใช้การเข้ารหัสลับสองชั้นกับกุญแจเซสชัน แต่ในเวอร์ชันที่ 2 ใช้หลักการของ Diffie-Hellman ซึ่งจะไม่มีการส่งกุญแจตัวจริงเข้าไปในเครือข่าย ทำให้ปลอดภัยมากขึ้น

4.2.3.4 Public Key Infrastructure (PKI)

ระบบที่ใช้กุญแจสาธารณะเราจะรู้อย่างไรว่า กุญแจสาธารณะที่ให้มาเป็นของเจ้าของตัวจริงไม่ใช่มีผู้อื่นมาแอบอ้าง เช่น อลิสต้องการติดต่อกับบ๊อบ แต่ทั้งคู่ไม่เคยติดต่อกันมาก่อน อลิสจะรู้ได้อย่างไรว่าคนที่ติดต่อด้วยเป็นบ๊อบ ดังนั้นจะต้องมีบุคคลกลางซึ่งทั้งอลิสและบ๊อบเชื่อใจ เช่นปีเตอร์ซึ่งปีเตอร์จะเซ็น (sign) ว่านี่คือกุญแจสาธารณะของบ๊อบแล้วส่งให้อลิส และเซ็นว่านี่คือกุญแจสาธารณะของอลิสแล้วส่งให้บ๊อบ

แบบจำลองดังกล่าวเป็นโครงสร้างที่เรียกว่า Public Key Infrastructure (PKI) ซึ่งเป็นแบบจำลองของการจัดการกุญแจสาธารณะที่เป็นที่นิยมมากในปัจจุบัน

4.2.3.5 การพิสูจน์ตน

ซีเคียวเชลเวอร์ชันที่ 2 จะยืดหยุ่นกว่าในเวอร์ชันที่ 1 เพราะเซิร์ฟเวอร์จะเป็นตัวบอกลำดับของอัลกอริทึมต่างๆ ต่างกับในเวอร์ชันที่ 1 ซึ่งถูกกำหนดตายตัวด้วยมาตรฐาน และนอกจากนี้ในเวอร์ชันที่สองยังสามารถกำหนดได้ละเอียดกว่า เช่น หากพิสูจน์ตนด้วยกุญแจสาธารณะสองครั้งแล้วยังไม่ผ่าน จะไม่ยอมให้พิสูจน์ตนด้วยวิธีอื่นอีกนอกจากใช้รหัสผ่าน

4.2.3.6 การเปลี่ยนกุญแจเซสชัน

ยิ่งมีข้อมูลที่เข้ารหัสลับกับกุญแจเดียวกันมากเท่าใด โอกาสแกะรหัสได้จะมากขึ้นตาม ดังนั้นการเปลี่ยนกุญแจเป็นระยะ จึงเป็นทางเลือกที่เหมาะสม ซึ่งสนับสนุนเฉพาะในเวอร์ชันที่ 2

4.3 เปรียบเทียบข้อแตกต่างของซีเคียวเชลเวอร์ชันที่ 1 และ 2

SSH-2	SSH-1
แยกเป็น 4 โพรโตคอล	มีโพรโตคอลเดียว
ใช้ MAC ในการตรวจสอบความถูกต้อง	ใช้ CRC-32 ซึ่งเสี่ยงต่อการถูกโจมตี
สนับสนุนการเปลี่ยนรหัสผ่าน	ไม่มีข้อมูล
จะมีกี่เซสชันต่อการเชื่อมต่อก็ได้	หนึ่งเซสชันหนึ่งการเชื่อมต่อ
สามารถเลือกได้ทั้งอัลกอริทึมเข้ารหัสลับ, บีบอัดข้อมูล และ MAC	เลือกได้เฉพาะอัลกอริทึมเข้ารหัสลับ
การเข้ารหัสลับ, MAC และการบีบอัดข้อมูล เลือกได้ในขาไปและขากลับ และกุญแจก็ไม่จำเป็นต้องเหมือนกัน	ทุกอย่างต้องเหมือนกัน ทั้งในขาไปและขากลับ
การเรียกชื่ออัลกอริทึมด้วยสตริงซึ่งสนับสนุนอัลกอริทึมที่พัฒนาขึ้นเองด้วย	กำหนดเป็นค่าตายตัวไม่สามารถเพิ่มได้
ใช้อัลกอริทึม Diffie-Hellman ในการหากุญแจเซสชัน ซึ่งขจัดความต้องการของกุญแจเซิร์ฟเวอร์ได้ ใช้แต่กุญแจโฮสต์เท่านั้น	ใช้การเข้ารหัสลับกุญแจเซสชันสองครั้งด้วยกุญแจเซิร์ฟเวอร์และกุญแจโฮสต์
มีการเปลี่ยนกุญแจเซสชันเป็นระยะ	ไม่มีข้อมูล

รูป 4-07 ความสัมพันธ์ของโพรโตคอลซีเคียวเชลเวอร์ชันที่ 2

บทที่ 5

การออกแบบ

โครงงานนี้ได้นำแนวคิดบางส่วนมาจากโปรแกรม NetCat และมีการเรียกใช้งานไลบรารี CryptLib ในส่วนของการ SSH-TRANS และ SSH-AUTH อันเนื่องมาจากซีเคียวเชลเวอร์ชันที่ 2 ยังคงเป็นมาตรฐานร่างซึ่งอาจมีการปรับปรุงเปลี่ยนแปลงได้ ดังนั้นหากพัฒนาขึ้นมาใช้เองอาจทำให้เกิดปัญหาในอนาคต หากมาตรฐานมีการเปลี่ยนแปลงต้องกลับมาแก้ไขซอร์สโคด แต่ถ้าหากเรียกใช้ไลบรารี เพียงแค่เปลี่ยนไฟล์ DLL ก็ใช้งานได้ ดังนั้นโครงงานนี้จึงเลือกการเรียกใช้ไลบรารีแทนการพัฒนาเอง

5.1 โปรแกรม NetCat

NetCat เป็นโปรแกรมอัตถประโยชน์ ที่ใช้ในการอ่านและเขียนข้อมูลผ่านเครือข่ายไอพี ซึ่งสามารถทำงานได้ทั้ง TCP และ UDP ซึ่งใช้ในการดีบักการทำงานของโพรโตคอลต่างๆ ทั้งบนยูนิกซ์และวินโดวส์

สำหรับโครงงานนี้นำการทำงานบางส่วนของ NetCat มาใช้ ได้แต่การเรียกใช้งานคำสั่งอื่น การรีไดเรก (Redirect) อินพุทและเอาท์พุท ซึ่งใน IsagSSH2 Server 2546 ใช้ในการเรียกคำสั่ง cmd.exe เมื่อใช้เป็นเชลและรีไดเรกอินพุทเอาท์พุทจาก cmd.exe ไปยังไคลเอนท์

5.2 ไลบรารี CryptLib

CryptLib เป็นเครื่องมือด้านความปลอดภัยสำเร็จรูป ซึ่งสนับสนุนการทำงานด้านความปลอดภัยหลายอย่าง เช่น SSH, SSL/TLS และ S/MIME เป็นต้น

โครงงานนี้ได้เรียกใช้งาน CryptLib ในส่วนของ SSH Transport Layer Protocol โดยทำหน้าที่ในส่วนของ SSH-TRANS ทั้งหมด ตั้งแต่การแลกเปลี่ยนกุญแจและอัลกอริทึมที่ใช้ จนถึงการเข้าและถอดรหัสลับ และใช้ SSH-AUTH เพื่อพิสูจน์ตนด้วยรหัสผ่าน

5.2.1 การคอมไพล์ CryptLib

CryptLib ที่ดาวน์โหลดได้จากอินเทอร์เน็ตจะมาในรูปแบบของซอร์สโค้ด ดังนั้นจึงต้องคอมไพล์ CryptLib เสียก่อน เมื่อคอมไพล์เสร็จก็จะได้ไฟล์ DLL (Dynamic Link Library)

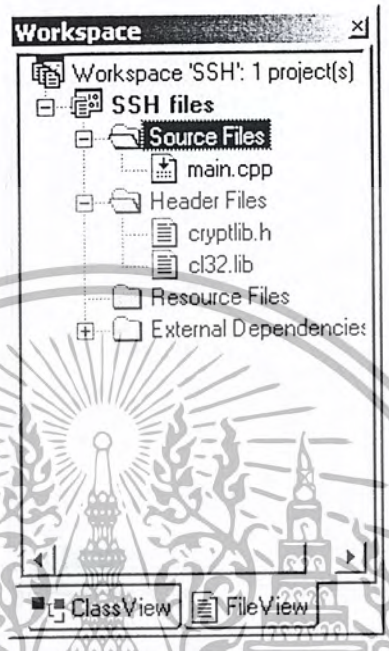
ไฟล์ DLL สร้างได้จากไฟล์ crypt32.dsp และ crypt32.dsw แล้วสั่ง Make ภายใต้ Microsoft Visual C++ 6.0 จากนั้นให้ใช้ test32.dsp เพื่อสร้างโปรแกรมคอนโซลทดสอบตนเอง เพื่อตรวจสอบว่า DLL ที่ได้ทำงานได้ถูกต้อง

เมื่อสร้าง DLL เสร็จจะได้ไฟล์ 3 ไฟล์คือ cl32.dll, cl32.exp และ cl32.lib โดยจะอยู่ในไดเรกทอรี [CryptLibRoot]\binary\

5.2.2 การใช้งาน CryptLib กับ Microsoft Visual C++

การใช้งาน CryptLib จำเป็นต้องใส่ไฟล์ 2 ไฟล์ คือ cryptlib.h และ cl32.lib ไว้ในเฮเดอร์ไฟล์ของโปรเจค และต้อง include cryptlib.h ในไฟล์ซอร์สโค้ด

เมื่อต้องการเรียกโปรแกรมที่เขียนขึ้นมาใช้งานต้องคัดลอกไฟล์ cl32.dll ไปไว้ในไดเรกทอรีเดียวกับไฟล์โปรแกรม หรือไว้ใน [SystemRoot]\system32\



รูป 5-01 เพิ่มไฟล์ cl32.lib ในโปรเจค

5.2.3 Initialisation

การเขียนโปรแกรมที่เรียกใช้งาน CryptLib จะต้องเรียก cryptInit() ก่อนเพื่อทำ initialisation และเรียก cryptEnd() หลังการใช้งานเพื่อทำลายตัวแปรและออบเจคต่างๆ โดย cryptInit อาจจะเรียก 1 ครั้งต่อ 1 thread ได้แต่ cryptEnd() ควรเรียกแค่ครั้งเดียวก่อนจบโปรแกรม ดังรูป 5-02

```
#include "cryptlib.h"
cryptInit();
/* Calls to cryptlib routines */
cryptEnd();
```

รูป 5-02 การเรียกใช้งาน cryptInit() และ cryptEnd()

5.2.4 Return Codes

ทุกฟังก์ชันของ CryptLib จะมีการคืนค่าเป็น int หากไม่มีข้อผิดพลาดเกิดขึ้นค่าที่คืนกลับจะเป็น CRYPT_OK หรือ 1 ดังรูป 5-03

```
int status;
status = cryptInit();
if( status != CRYPT_OK )
      /* cryptlib initialisation
failed */;
/* Calls to cryptlib routines */
status = cryptEnd();
if( status != CRYPT_OK )
      /* cryptlib shutdown failed */;
```

รูป 5-03 การตรวจสอบค่าผิดพลาดจากค่าที่คืนจากฟังก์ชัน

5.2.5 การสร้างเซสชันของ CryptLib

การเรียกใช้งานเซสชันใน CryptLib เป็นไปตามนี้

1 สร้างเซสชัน

2 ใส่เซิร์ฟเวอร์คีย์

3 แอคทีฟเซสชัน

4 รอข้อมูลจากไคลเอนท์

5 ส่งข้อมูลกลับไปที่ไคลเอนท์

6 ทำลายเซสชัน

5.2.6 การสร้างกุญแจส่วนตัว และกุญแจสาธารณะ

ทำได้โดยประกาศตัวแปรเป็น CRYPT_CONTEXT จากนั้นเรียกฟังก์ชัน cryptCreateContext() และฟังก์ชัน cryptSetAttributeString() เพื่อกำหนดค่าเริ่มต้นก่อนการสร้างกุญแจ จากนั้นเรียกฟังก์ชัน cryptGenerateKey() เพื่อสร้างกุญแจ

```
CRYPT_CONTEXT privKeyContext;
cryptCreateContext( &privKeyContext, CRYPT_UNUSED, ↵
      CRYPT_ALGO_DSA );
cryptSetAttributeString( privKeyContext, ↵
      CRYPT_CTXINFO_LABEL, "Private key", 11 );
cryptGenerateKey( privKeyContext );
```

รูป 5-04 การสร้างกุญแจส่วนตัว และกุญแจสาธารณะ

ขั้นตอนนี้ใช้เวลาค่อนข้างนานโดยเฉพาะอย่างยิ่งเมื่อใช้กับเครื่องรุ่นเก่า

กุญแจที่ได้ในขั้นตอนนี้จะได้มาเป็นคู่ คือ กุญแจส่วนตัว และกุญแจสาธารณะ

ในฟังก์ชัน cryptSetAttributeString() ค่า 11 หมายถึงความยาวสตริง “Private key”

5.2.7 การเก็บกุญแจส่วนตัว และกุญแจสาธารณะลงในดิสก์

CryptLib สามารถเก็บรักษากุญแจได้หลายแบบ หนึ่งในวิธีเหล่านั้น ซึ่งเป็นวิธีที่ IsagSSH2 Server 2546 เลือกใช้คือ การเก็บเป็นไฟล์ลงในดิสก์ โดยไฟล์ที่เก็บนั้นจะมีระดับการอนุญาต (permission) ตามระดับของผู้ที่สร้างไฟล์ ตัวอย่างแสดงอยู่ในรูป 5-05

```
CRYPT_KEYSET cryptKeyset;
cryptKeysetOpen( &cryptKeyset, CRYPT_UNUSED, ↵
      CRYPT_KEYSET_FILE, "host.key", CRYPT_KEYOPT_CREATE );
cryptAddPrivateKey( cryptKeyset, privKeyContext, "Isag" );
cryptKeysetClose(cryptKeyset);
```

รูป 5-05 การเก็บกุญแจส่วนตัว และกุญแจสาธารณะลงในดิสก์

สำหรับไฟล์ที่เก็บอยู่บนดิสก์จะมีการเข้ารหัสด้วยรหัสผ่านเพื่อป้องการการอ่านโดยไม่ได้รับอนุญาต โดยในรูป 5-05 รหัสผ่านที่ใช้คือ "Isag"

5.2.8 การรับและส่งข้อมูลผ่านทางเซสชันของ CryptLib

การส่งข้อมูลผ่านเซสชันสามารถทำได้โดยใช้ฟังก์ชัน cryptPushData() และการรับข้อมูลผ่านเซสชันสามารถทำได้โดยใช้ฟังก์ชัน cryptPopData() โดยข้อมูลนี้จะถูกเข้ารหัสลับเวลารับและส่ง

5.3 เอพีไอของวินโดวส์

วินโดวส์ได้จัดเอพีไอ (Application Program Interface : API) ที่ใช้ติดต่อกับระบบปฏิบัติการ และ IsagSSH2 Server 2546 ได้เรียกใช้เอพีไอบางตัวของวินโดวส์ สำหรับเอพีไอที่น่าสนใจ ได้แก่

CreateProcessAsUser

ฟังก์ชันนี้จะเรียกโพรเซสที่ระบุขึ้นมาทำงาน ด้วยสิทธิของผู้ใช้ตามที่ระบุด้วย token

LogonUser

ฟังก์ชันนี้จะรับชื่อผู้ใช้ โดเมน และรหัสผ่าน หากการผ่านการตรวจสอบ จะให้ token ที่มีสิทธิ์ของผู้ใช้คนนั้น เพื่อเอาไปใช้ในฟังก์ชันอื่น เช่น CreateProcessAsUser()

WaitForMultipleObjects

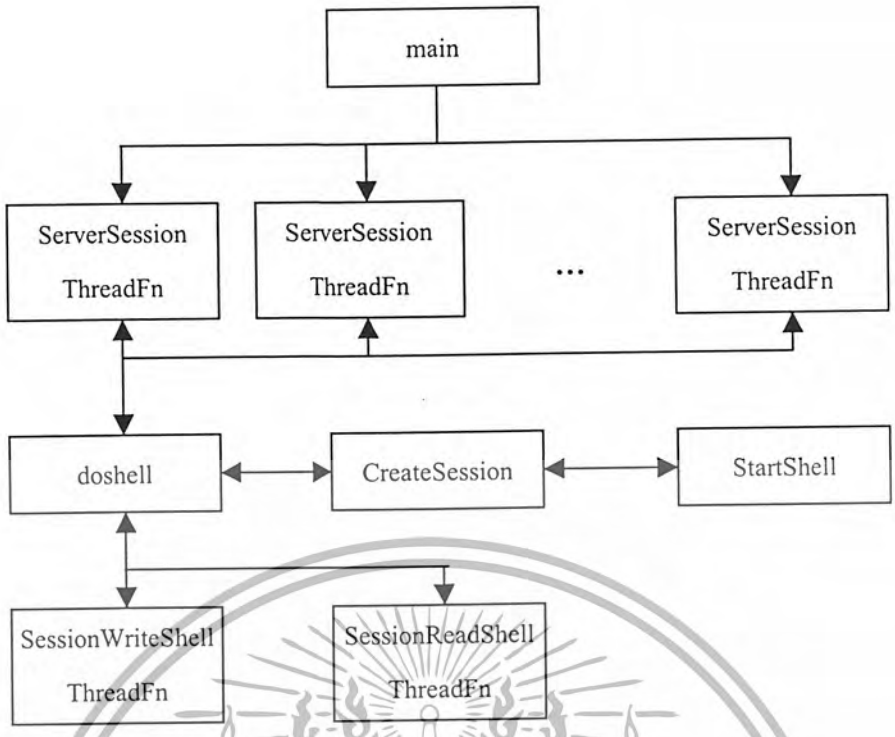
เป็นฟังก์ที่ใช้ตรวจสอบว่า thread ไหนจบการทำงาน หรือใช้ตรวจสอบว่าทุก thread จบการทำงานแล้ว

5.4 โครงสร้างของโปรแกรม IsagSSH2 Server 2546

รูป 5-06 จะแสดงโครงสร้างของโปรแกรม IsagSSH2 Server 2546 โดยแบ่งเป็นส่วนๆ ดังนี้

main

หน้าที่: สร้าง thread ขึ้นมารองรับการติดต่อจากไคลเอนท์ โดยจำนวนของ thread เท่ากับจำนวนการเชื่อมต่อสูงสุดที่รับได้ในเวลาเดียวกัน (MAX_CONN) ซึ่งถูกกำหนดไว้ในไฟล์ชื่อ host.config และเมื่อ thread ใดจบการทำงานก็จะสร้าง thread ใหม่ขึ้นมาทำงานแทน



รูป 5-06 โครงสร้างของ IsagSSH2 Server 2546

ServerSessionThreadFn

ข้อมูลเข้า: Private host key

หน้าที่หลัก: รอรับการติดต่อจากไคลเอนท์ โดยใช้ 1 thread ต่อ 1 การเชื่อมต่อและตรวจสอบชื่อผู้ใช้และรหัสผ่าน โดยใช้ฟังก์ชัน LogonUser ถ้าถูกต้องจะได้ primary token ซึ่งแสดงสิทธิ์ของผู้ใช้คนนั้น นำไปทำการเรียกฟังก์ชัน doshell เพื่อทำงานเกี่ยวกับ shell process ต่อไป

doshell

ข้อมูลเข้า: cryptSession, ตัวชี้ของ primary token

ข้อมูลออก: ตัวแปร Boolean แสดงว่าการเรียกเชลโพรเซส (shell process) สำเร็จหรือไม่

หน้าที่หลัก: เป็นฟังก์ชันหลักในการสร้างเชลโพรเซสและทำการสร้าง thread ที่ใช้ถ่ายโอนข้อมูลระหว่างเชลโพรเซสและไคลเอนท์

CreateSession

ข้อมูลเข้า: ตัวชี้ของ primary token

ข้อมูลออก: ตัวชี้ของ struct แบบ SESSION_DATA

หน้าที่หลัก: สร้าง pipe เพื่อใช้ถ่ายโอนข้อมูลจาก stdin stdout ของเชลโพรเซสจากนั้นจึงเรียกฟังก์ชัน StartShell เพื่อทำการสร้างเชลโพรเซส

StartShell

ข้อมูลเข้า: ตัวชี้ของ primary token, handle ของ stdin และ stdout ของ shell process

ข้อมูลออก: handle ของเชลโพรเซส

หน้าที่หลัก: ทำการสร้างเชลโพรเซสโดยใช้ฟังก์ชัน CreateProcessAsUser (ใช้ primary token ในการกำหนดสิทธิ์ของโพรเซส) ส่วน stderr ของเชลโพรเซสจะใช้ handle ร่วมกับ stdout โดยใช้ฟังก์ชัน DuplicateHandle

SessionWriteShellThreadFn

ข้อมูลเข้า: ตัวชี้ของ struct แบบ SESSION_DATA

หน้าที่หลัก: ทำหน้าที่อ่านข้อมูลจากไคลเอนท์ด้วยฟังก์ชัน cryptPopData แล้วนำไปเก็บใส่บัฟเฟอร์ ทำการแปลงข้อมูลให้อยู่ในรูปที่พร้อมแสดงผล จากนั้นจึงส่งไปให้ไคลเอนท์ด้วยฟังก์ชัน cryptPushData เพื่อแสดงผลที่ฝั่งไคลเอนท์ และเมื่อไคลเอนท์กดปุ่ม Enter จึงค่อยนำไปเขียนลง stdin ของเชลโพรเซส

SessionReadShellThreadFn

ข้อมูลเข้า: ตัวชี้ของ struct แบบ SESSION_DATA

หน้าที่หลัก: อ่านข้อมูลจาก stdout (รวมทั้ง stderr ซึ่งใช้ handle เดียวกัน) ของเชลโพรเซส แล้วไปนำเก็บใส่บัฟเฟอร์ ทำการแปลงข้อมูลให้อยู่ในรูปแบบมาตรฐาน (แทนที่ LF ตัวโดดๆด้วย CR-LF) จึงค่อยส่งข้อมูลไปให้ไคลเอนท์ด้วยฟังก์ชัน cryptPushData

บทที่ 6

การทดสอบและผลการทดสอบ

6.1 จุดประสงค์การทดสอบ

1. ทดสอบการทำงานว่าสามารถทำงานได้ถูกต้องตามโพรโตคอลซีเคียวเชลล์
2. ทดสอบการพิสูจน์ตนว่าสามารถใช้กับผู้ใช้ในวินโดวส์ 2000 ได้ตามปกติ
3. ทดสอบการทำงานเมื่อมีเครื่องไคลเอนท์หลายเครื่องติดต่อเข้ามาพร้อมกัน

6.2 สภาพแวดล้อมและโปรแกรมที่ใช้ในการทดสอบ

6.2.1 โปรแกรมที่ใช้ในการทดสอบ

1. IsagSSH2 Server 2546
2. SecureCRT
3. Ethereal

6.2.2 การติดตั้งและปรับแต่ง IsagSSH2 Server 2546

6.2.2.1 การสร้างโฮสต์คีย์ (Host Key)

โฮสต์คีย์จะเข้ารหัสลับและเก็บอยู่ในไฟล์ชื่อ host.key ในไดเรกทอรีเดียวกับโปรแกรม แต่สามารถเลือกได้ว่าจะใช้ชื่ออะไร พาธไหน

หากยังไม่มีโฮสต์คีย์ โปรแกรมจะสร้างให้อัตโนมัติ หากต้องการเปลี่ยนโฮสต์คีย์ ให้ลบโฮสต์คีย์ไฟล์เก่าเพื่อที่โปรแกรมจะได้สร้างให้ใหม่

6.2.2.2 การติดตั้งโปรแกรม

เนื่องจากโปรแกรมนี้มีการเปลี่ยน privilege ของโปรเซส เพื่อใช้เรียน cmd.exe จึงต้องมีการแก้ไขในตัว Security Policy ทำได้โดย

1. เปิด Local Security Policy (Start->Setting->Control Panel->Administrative Tools->Local Security Policy)
2. เพิ่มชื่อผู้ใช้ที่เรียกโปรแกรม IsagSSH2 Server 2546 ลงใน Replace a process level token (Security Setting->Local Policies->User Rights Assignment)
3. คัดลอกโปรแกรมทั้งหมดลงในไดเรกทอรีที่ต้องการ
4. เรียกใช้งานโปรแกรม

หมายเหตุ: หากเครื่องที่ทำงานอยู่ใน Domain หรือเป็น Domain Controller ต้องเข้าไปแก้ใน Domain Security Policy หรือ Domain Controller Security Policy ตามลำดับ เพราะค่าที่มีผลบังคับใช้คือค่าที่กำหนดไว้ในระดับที่สูงกว่า

6.2.2.3 การปรับแต่งโปรแกรม IsagSSH2 Server 2546

การปรับแต่งโปรแกรมจะใช้เป็นเท็กซ์ไฟล์ เช่นเดียวกับระบบยูนิกซ์ โดยค่าที่สามารถปรับแต่งได้มีดังนี้

MAX_CONN

เป็นจำนวนการเชื่อมต่อจากไคลเอนท์มากที่สุด ในเวลาเดียวกัน ที่สามารถทำได้ ค่าปริยายคือ 5

SERVER_PORT

เป็นหมายเลขทีซีพีพอร์ตของโปรแกรมที่จะรอรับการเชื่อมต่อ จากประกาศของ IANA กำหนดให้ซีเคียวเชลเซิร์ฟเวอร์ใช้งานพอร์ตหมายเลข 22

HKEY_FILE

เป็นไฟล์ที่เก็บกุญแจโฮสต์ โดยระบุเป็นพาธแบบเต็ม หรือ 0 ซึ่งหมายถึงไดเรกทอรีเดียวกับโปรแกรม และใช้ชื่อไฟล์ว่า host.key

LOG_FILE

เป็นไฟล์ที่เก็บ log ของโปรแกรม โดยระบุเป็นพาธแบบเต็ม หรือ 0 ซึ่งหมายถึงไดเรกทอรีเดียวกับโปรแกรม และใช้ชื่อไฟล์ว่า host.log

6.2.2.4 การเรียกใช้โปรแกรม

ดับเบิ้ลคลิ๊กที่ไฟล์ ssh2d.exe เพื่อใช้งานโปรแกรม

6.3 วิธีทดสอบและผลการทดทดสอบ

6.3.1 การทดสอบการเข้ารหัสลับของโปรแกรม

6.3.1.1 วิธีการทดสอบ

1. เปิดโปรแกรม Ethereal เพื่อดักจับข้อมูล
2. เรียกใช้งาน IsagSSH2 Server 2546
3. ให้ SecureCRT เชื่อมต่อมาที่เซิร์ฟเวอร์

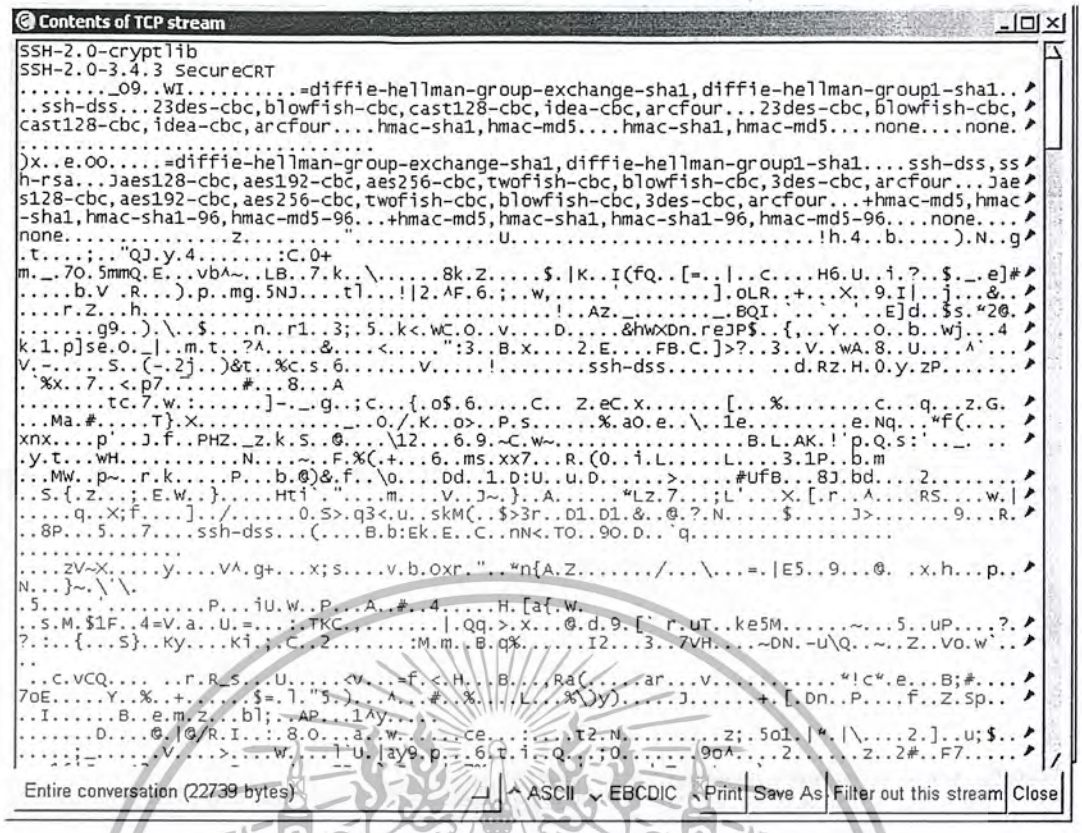
6.3.1.2 ผลการทดสอบ

ข้อมูลที่ดักจับได้ ไม่สามารถอ่านและทำความเข้าใจได้ ดังรูปที่ 6-1

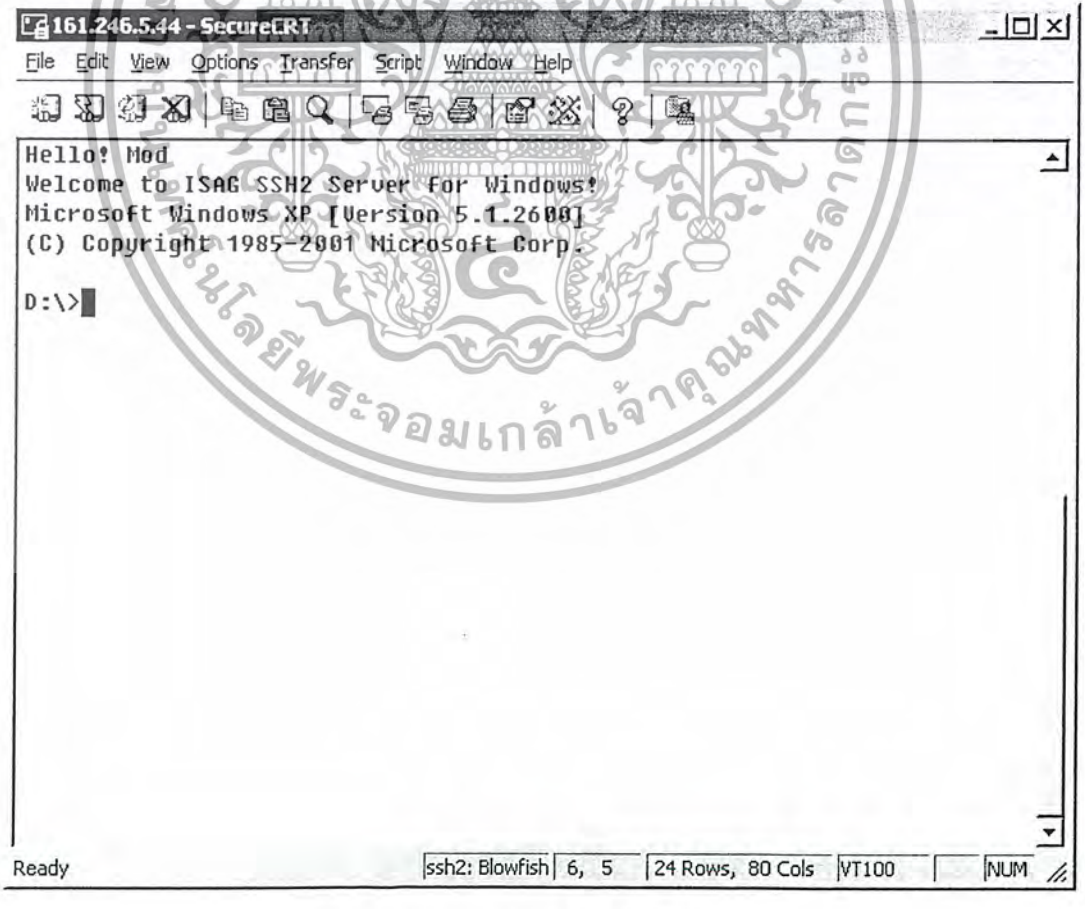
6.3.2 การทดสอบการพิสูจน์ตนด้วยบัญชีผู้ใช้

6.3.2.1 วิธีการทดสอบ

1. ทดสอบกับบัญชีผู้ใช้ปกติ
2. ทดสอบกับบัญชีผู้ใช้ที่รหัสผ่านไม่ถูกต้อง
3. ทดสอบกับบัญชีผู้ใช้ที่ถูกยกเลิก (disable)
4. ทดสอบกับบัญชีผู้ใช้ที่ไม่มี



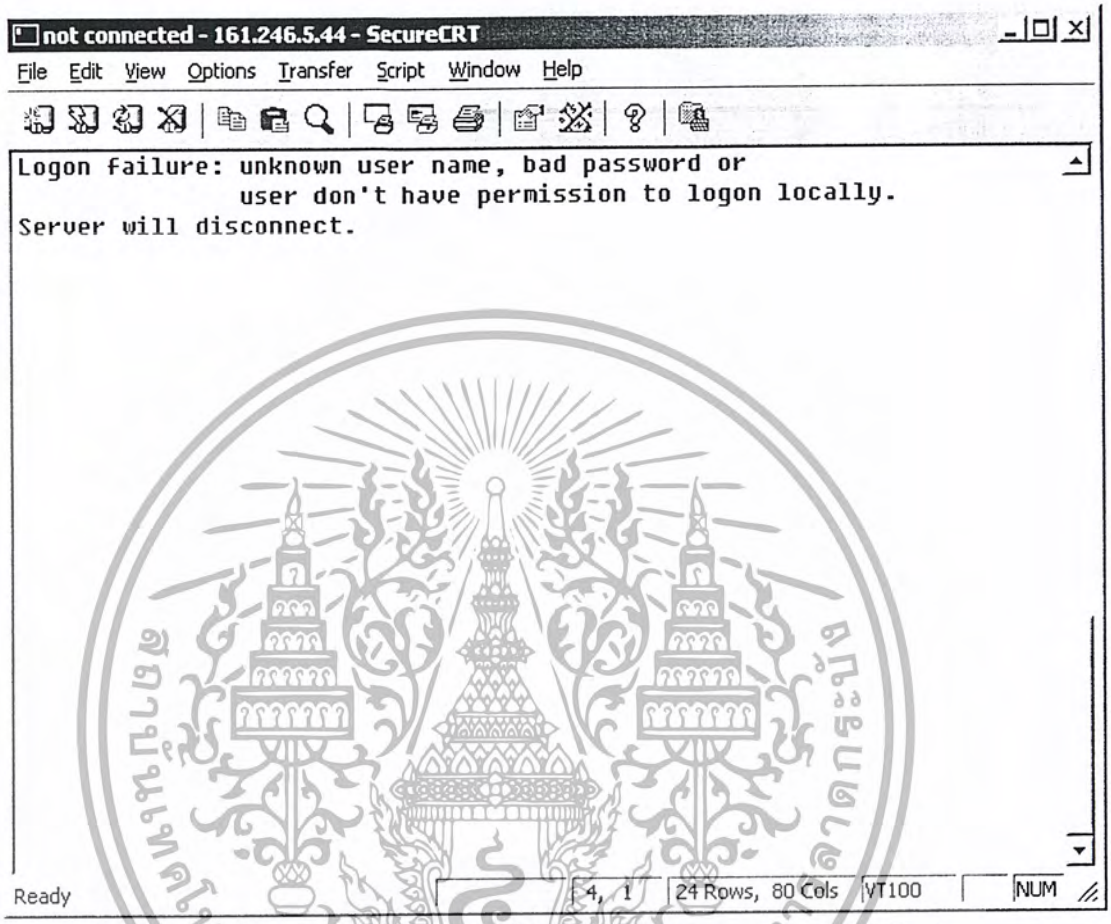
รูป 6-1 ผลทดสอบการเข้ารหัสลับ



รูป 6-02 ผลการทดสอบบัญชีผู้ใช้ปกติ

6.3.2.2 ผลการทดสอบ

หากมีบัญชีผู้ใช้ ป้อนรหัสผ่านที่ถูกต้อง และไม่ถูกยกเลิกสามารถใช้งานได้ตามปกติ ดังรูป 6-02 แต่ถ้าไม่ได้เซิร์ฟเวอร์จะแจ้งข้อความผิดพลาดและตัดการเชื่อมต่อทันที ดังรูป 6-03



รูป 6-03 ผลการทดสอบบัญชีผู้ใช้ที่ป้อนรหัสผ่านผิด ถูกยกเลิก หรือ ไม่มีอยู่

6.3.3 การทดสอบการเปลี่ยนที่ซีพีพอร์ต

6.3.3.1 วิธีการทดสอบ

แก้ไขไฟล์ host.config โดยเปลี่ยน SERVER_PORT 22 เป็นหมายเลขที่ต้องการ

1. เป็นพอร์ต 23 ซึ่งไม่มีโปรแกรมใดจองอยู่
2. เป็นพอร์ต 21 ซึ่งมีโปรแกรมอื่นใช้อยู่
3. เป็นพอร์ต 65536 ซึ่งอยู่นอกช่วงที่เป็นไปได้
4. ไม่ระบุพอร์ต

6.3.3.2 ผลการทดสอบ

1. ใช้งานได้ตามปกติ
2. แจ้งข้อความผิดพลาด ดังรูป 6-04
3. แจ้งข้อความผิดพลาด ดังรูป 6-05

4. ใช้งานได้ตามปกติ โดยใช้พอร์ต 22

```
Port 80 already in used.
Please select new port in file "host.config".
```

รูป 6-04 ข้อความผิดพลาด เนื่องจากมีโปรแกรมอื่นใช้พอร์ตอยู่

```
Invalid port number: 67000 (1-65535 Default: 22).
Please select new port in file "host.config".
```

รูป 6-05 ข้อความผิดพลาด เนื่องจากมีพอร์ตอยู่นอกช่วงที่เป็นไปได้

บทที่ 7
บทวิจารณ์และสรุป

โครงงานนี้ได้พัฒนาโปรแกรมเพื่อเรียกใช้งานอินเทอร์แอกทีฟเซสชัน (Interactive Session) ตามมาตรฐานของโพรโตคอลซีเคียวเชลเวอร์ชันที่ 2 จากการศึกษาและพัฒนามีข้อสรุปดังนี้

7.1 บทวิจารณ์และสรุปโพรโตคอลซีเคียวเชล 2

มีการแบ่งโพรโตคอลออกเป็นหลายส่วนเพื่อความสะดวกในการพัฒนา แต่เนื่องจากปัจจุบันยังเป็นมาตรฐานร่าง (Draft Standard) จึงอาจมีการเปลี่ยนแปลงของข้อกำหนดต่างๆ ได้ ดังนั้นโปรแกรมที่ทำงานในมาตรฐานนี้อาจมีปัญหาในเรื่องความเข้ากันได้ (Compatible) สำหรับข้อดีของโพรโตคอล คือ

- ไม่สามารถปลอมแปลงไอพีได้ เพราะเข้ารหัสด้วยโฮสต์คีย์ และใช้ข้อมูลที่สุ่มขึ้นในการป้องกันการปลอมแปลง
- ชื่อบัญชีผู้ใช้และรหัสผ่านจะถูกเข้ารหัสด้วยการเข้ารหัสลับแบบกุญแจเดี่ยวเพื่อความปลอดภัยและความรวดเร็ว
- กุญแจเดี่ยวจะถูกเข้ารหัสด้วยการเข้ารหัสลับแบบกุญแจสาธารณะก่อนส่ง
- การโจมตีของบุคคลระหว่างกลาง (Man in The Middle Attack) ทำได้ยากขึ้น เพราะใช้ HMAC ที่มีความปลอดภัยมากกว่า CRC ที่ใช้ในเวอร์ชันแรก

7.2 บทวิจารณ์และสรุปโปรแกรม IsagSSH2 Server 2546

การทำงานในส่วนของ SSH-TRANS และ SSH-AUTH บางส่วนได้เรียกใช้งานไลบรารี CryptLib เวอร์ชัน 3.1 ดังนั้นหากมีการเปลี่ยนแปลงมาตราฐานของโพรโตคอลซีเคียวเชลเวอร์ชัน 2 หรือออกมาเป็นมาตรฐานจริง ก็เปลี่ยนแค่ไลบรารี ซึ่งเป็นไฟล์ dll เพียงตัวเดียว ซึ่งสะดวกเป็นอย่างมาก

สำหรับการตรวจสอบผู้ใช้ได้ใช้เอพีไอ (API) ของวินโดวส์ จึงสามารถใช้บัญชีผู้ใช้เดียวกับวินโดวส์ได้

7.3 ข้อจำกัดของโปรแกรม IsagSSH2 Server 2546

ปัญหาและข้อจำกัดของโปรแกรมที่พบมีดังนี้

- หากใช้งานคำสั่งที่มีผลลัพธ์จำนวนมาก เช่น dir/s โปรแกรมจะเกิดข้อผิดพลาดขึ้น
- หากใช้คำสั่งที่มีการส่วนติดต่อผู้ใช้แบบกราฟฟิก (GUI) อย่าง edit.com หน้าจอโปรแกรม edit จะปรากฏที่เซิร์ฟเวอร์ ไม่สามารถส่งไปที่ไคลเอนท์ได้ และต้องสั่งปิดจากเซิร์ฟเวอร์เท่านั้น
- ไม่สามารถใช้งานคำสั่งที่ต้องรอรับอินพุทอีกครั้งได้ เช่น dir/p,. runas เป็นต้น หากใช้งานอินพุทที่รอรับจะเป็น NULL
- ยังไม่มีการทดสอบบัญชีผู้ใช้ของเครื่องที่อยู่ในโดเมนหรือเป็นโดเมนคอนโทรเลอร์

7.4 แนวทางการพัฒนาต่อในอนาคต

- ทำเป็น System Service เพื่อความสะดวกในการใช้งาน
- เพิ่มความสามารถส่วนของ TCP Port Forwarding
- เพิ่มความสามารถส่วนของ Secure File Transfer (SFTP)
- เพิ่มออปชันในไฟล์คอนฟิกให้ยืดหยุ่นมากขึ้น

ภาคผนวก ก.

Microsoft Visual C++

Microsoft Visual C++ เป็นโปรแกรมที่สามารถใช้เป็นเครื่องมือ (Tool) ในการพัฒนาโปรแกรมประยุกต์บนระบบปฏิบัติการวินโดวส์ (Microsoft Windows) ที่มีความสามารถในการจัดการและควบคุมการติดต่อหรือใช้งานทั้งด้านซอฟต์แวร์และทางด้านฮาร์ดแวร์สูง ซึ่งตัวโปรแกรมเองได้รับการพัฒนาความยืดหยุ่นและความมีประสิทธิภาพสูงนี้มาจากภาษา C++

โดยภาษา C++ มีรากฐานมาจากภาษา C และเพิ่มความสามารถเรื่องการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming : OOP) ซึ่งเป็นวิธีการเขียนโปรแกรม โดยอาศัยแนวคิดของวัตถุ (Object) ชิ้นหนึ่ง ต่างจากแนวคิดเดิมที่เป็นแบบโครงสร้าง (Structure) โดยการเขียนโปรแกรมเชิงวัตถุนี้มีความสามารถในการปกป้องข้อมูล โดยสามารถซ่อนรายละเอียด (encapsulation) และการสืบทอดคุณสมบัติ (inheritance) ของวัตถุตัวอื่นๆ ซึ่งทำให้การเขียนโปรแกรมมีความง่ายและรวดเร็วขึ้นมาก

- คลาส (Class) คือ การรวมคุณลักษณะและการใช้งานของวัตถุอย่างน้อยหนึ่งอย่างมาไว้ในกลุ่มเดียวกัน

- ออบเจ็กต์ (Object) คือ วัตถุที่เป็นตัวแปรของคลาส เป็นรูปแบบของคลาสที่มีตัวตน ที่เราสามารถนำไปใช้งานได้

- การสืบทอดของคลาส (Inheritance) คือ การที่คลาสแต่ละคลาสสามารถเป็นคลาสแม่ (Super Class) ได้ โดยจะสืบทอดมาเป็นคลาสใหม่ได้ ซึ่งคลาสใหม่ที่สืบทอดมานี้จะยังคงมีคุณสมบัติเหมือนกับคลาสแม่ทุกประการและเราสามารถกำหนดคุณสมบัติใหม่เพิ่มเติมลงไปได้อีกเพื่อความเหมาะสมในการใช้งาน

โปรแกรม Microsoft Visual C++ ยังสนับสนุนการพัฒนาโปรแกรมประยุกต์อื่นๆในหลายด้านด้วยกัน ไม่ว่าจะเป็นโปรแกรมประยุกต์ที่ทำงานทั่วไป เช่น การคำนวณง่ายๆ การเขียนรูปต่างๆ เป็นต้น การเขียนโปรแกรมระบบจัดการฐานข้อมูล (DBMS : Database Management System) หรือแม้กระทั่งการเขียนโปรแกรมที่ทำงานด้วยระบบมัลติมีเดีย (Multimedia Application)

ขั้นตอนแรกในการสร้างโปรแกรมประยุกต์โดยใช้โปรแกรม Microsoft Visual C++ คือ การสร้างโปรเจกต์เวิร์กสเปซ (Project Workspace) ขึ้นมาใหม่ ซึ่งเป็นการกำหนดพื้นที่ในการเก็บโปรเจกต์ (Project) หรือเก็บโปรแกรมที่เราต้องการสร้าง และใช้ในการกำหนดตัวเลือกต่างๆของโปรแกรมที่เราต้องการสร้าง เช่น เก็บรูปภาพ การกำหนดลักษณะของโปรแกรมที่สร้าง หรือข้อกำหนดต่างๆเป็นต้น

การใช้งานโปรเจกต์ของ Microsoft Visual C++ มีลักษณะการใช้งานเหมือนกับโปรเจกต์ไฟล์ทั่วไป คือ เราสามารถเปิด-ปิด (Open-Close) เซฟ (Save) หรือลบ (Delete) ไฟล์ที่ใช้งานได้ สามารถคอมไพล์ (Compile) รวมทั้งการดีบัก (Debug) โปรแกรมเพื่อหาจุดผิดของโปรแกรมได้

เราจะเห็นได้ว่าในการเขียนโปรแกรมประยุกต์บนระบบปฏิบัติการดอส (Dos) เราสามารถเขียนโปรแกรมใหญ่ๆได้โดยใช้ไฟล์เดียวในการเก็บข้อมูลทุกอย่าง แต่ในการเขียนโปรแกรมประยุกต์บนระบบ

ปฏิบัติการวินโดว์นั้นเรามักจะต้องใช้ส่วนประกอบต่างๆมากมาย เช่น รูปภาพ หรือเมนูบาร์ (Menu Bar) เป็นต้น และใช้ไฟล์ร่วมกันหลายๆไฟล์ เช่น ไฟล์ที่ทำหน้าที่ในการเปิดรูปภาพ หรือไฟล์ที่ทำการแบ่งส่วนของรูปภาพเป็นต้น เพื่อให้เป็นสัดส่วนและสามารถแก้ไขโปรแกรมนี้ได้ง่ายยิ่งขึ้น

โปรเจกต์ไฟล์ของ Microsoft Visual C++ เวอร์ชัน 6 ใช้นามสกุล dsw ซึ่งทำหน้าที่เป็นไฟล์ที่เก็บตัวเลือกต่างๆของโปรเจกต์ดังที่กล่าวไปแล้ว และเราสามารถโหลดโปรเจกต์ไฟล์ที่เขียนด้วย Microsoft Visual C++ เวอร์ชันที่ต่ำกว่านี้ได้ เช่น mak หรือ mdp เป็นต้น

ประโยชน์ของโปรเจกต์ไฟล์สามารถสรุปได้ดังนี้คือ

1. โปรเจกต์ไฟล์จะเก็บรายชื่อของไฟล์ที่เป็นซอร์สโค้ด (Source Code) โปรแกรมทั้งหมดที่ใช้ร่วมกันในโปรเจกต์ เช่น ซอร์สโปรแกรมนามสกุล h หรือ นามสกุล cpp รวมทั้งไฟล์ฐานข้อมูลโปรแกรมที่ใช้ในคลาสวิซาร์ด (Class Wizard) เป็นต้น

2. โปรเจกต์ไฟล์จะเก็บค่าตัวเลือกสำหรับการคอมไพล์และลิงค์ (Link) กับไลบรารี (Library) ใดๆ หรือมีการสร้างส่วนประกอบ (Component) อื่นๆอีกหรือไม่ เช่น ส่วนประกอบในการดีบัก

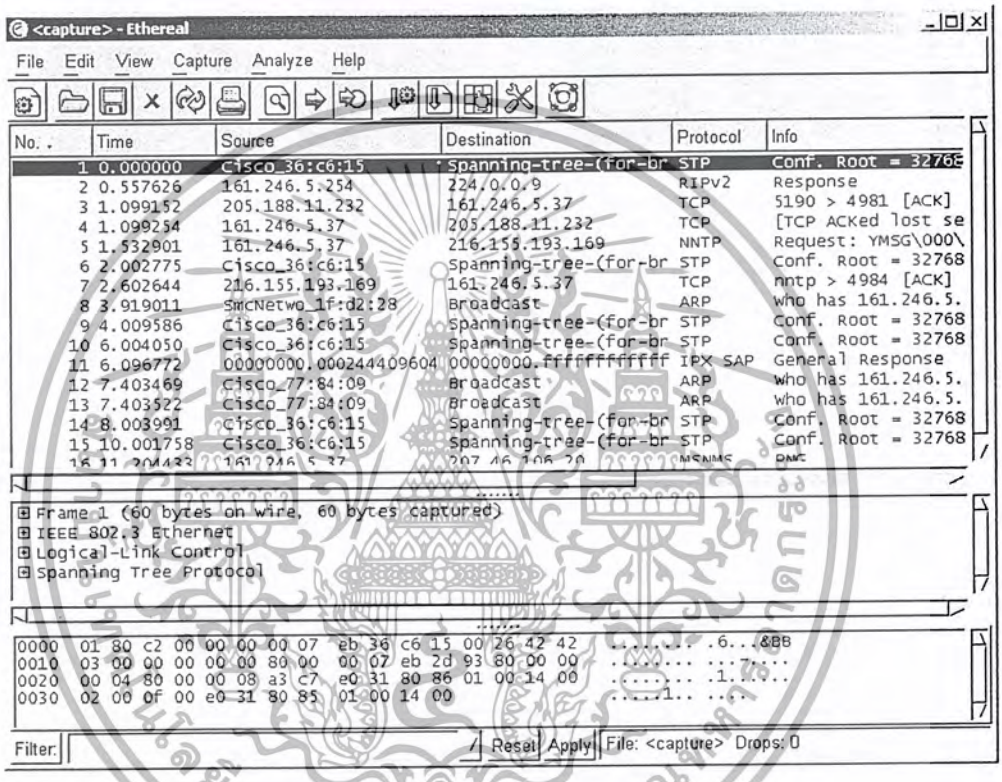
3. โปรเจกต์จะเก็บค่าตัวเลือกที่แสดงว่าโปรเจกต์นี้เป็นโปรเจกต์แบบใดเมื่อทำการคอมไพล์ เช่น เป็นวินโดว์แอปพิเคชั่น (Windows Application) โดยมีนามสกุลเป็น exe หรือเป็นพวกไดนามิกลิงค์ไลบรารี (Dynamic-Ling Library) โดยมีนามสกุลเป็น dll เป็นต้น

ภาคผนวก ข.

Ethereal

Ethereal เป็นโปรแกรมดักจับแพ็กเก็ตตัวหนึ่งที่ได้รับความนิยมมาก เนื่องมาจากเป็นโปรแกรมโอเพ่นซอร์ส ทั้งยังสนับสนุนในหลาย ๆ แพลตฟอร์ม ทั้งยูนิกส์ ลีนุกซ์ และวินโดวส์

การทำงานของโปรแกรม Ethereal จำเป็นต้องอาศัยไลบรารีในยูนิกซ์ชื่อว่า libpcap ซึ่งมีการแปลงมาใช้กับวินโดวส์ชื่อว่า WinPcap เพื่อใช้อ่านข้อมูลจากเครือข่าย



รูป ข-01 โปรแกรม Ethereal

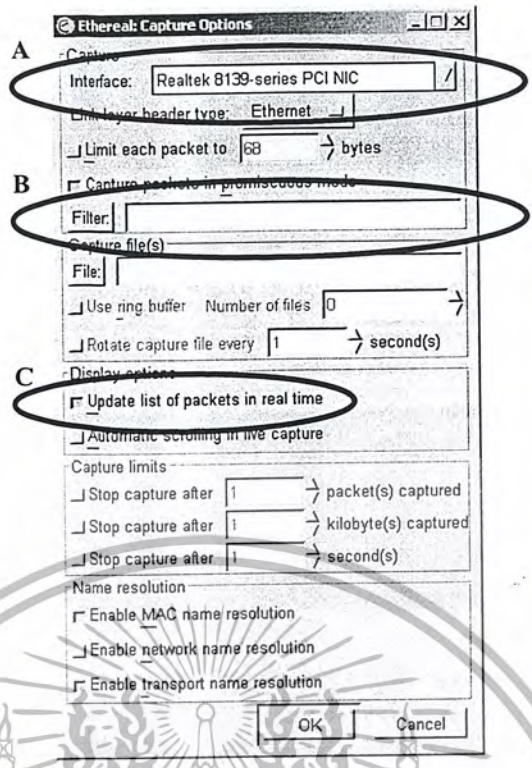
การดักจับแพ็กเก็ตเบื้องต้น

การใช้งาน Ethereal เบื้องต้น ทำได้โดยเลือกที่ Capture -> Start จากนั้นจะปรากฏกรอบสนทนา (Dialog Box) ขึ้นมาถาม ดังรูป ข-02

ตำแหน่ง A เป็นลิสต์ของเน็ตเวิร์คอินเทอร์เฟสที่จะดักจับ

ตำแหน่ง B เป็นเงื่อนไขที่ใช้ในการกรองแพ็กเก็ต ถ้าไม่ใส่หมายถึงทุกอย่าง ซึ่งจะอธิบายถัดไป

ตำแหน่ง C เป็น checked box สำหรับบอกว่าให้อัพเดทแพ็กเก็ตที่ดักจับได้ในทันที หรือว่าแสดงพร้อมกันทีเดียวหลังจากหยุดดักจับ



รูป ข-02 กรอบสนทนาการดักจับแพ็กเก็ต

การกำหนดเงื่อนไขที่ใช้กรองแพ็กเก็ต

คำที่สามารถนำมาใช้ในการกรองที่ใช้บ่อย ได้แก่

AND, OR, NOT	เป็นโอเปอร์เตอร์ทางตรรกะทั่วไป
TCP PORT *port* number	กรองเฉพาะทีซีพีพอร์ตที่กำหนด
HOST *hostname or ip address*	กรองเฉพาะเครื่องที่มีชื่อตามที่ระบุไว้

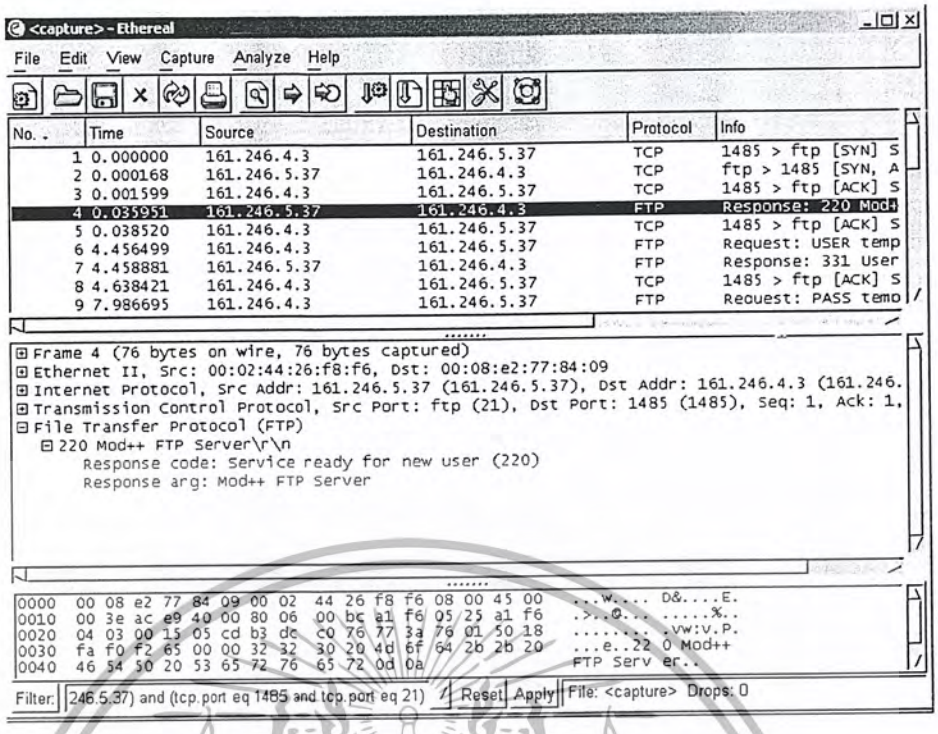
การใช้งานสามารถมาผสมกัน โดยสามารถใช้วงเล็บในการจัดกลุ่มได้เช่น

```
NOT TCP PORT 80
(HOST 161.246.4.3) AND (TCP PORT 80)
```

รูป ข-03 ตัวอย่างเงื่อนไขที่ใช้กรองแพ็กเก็ต

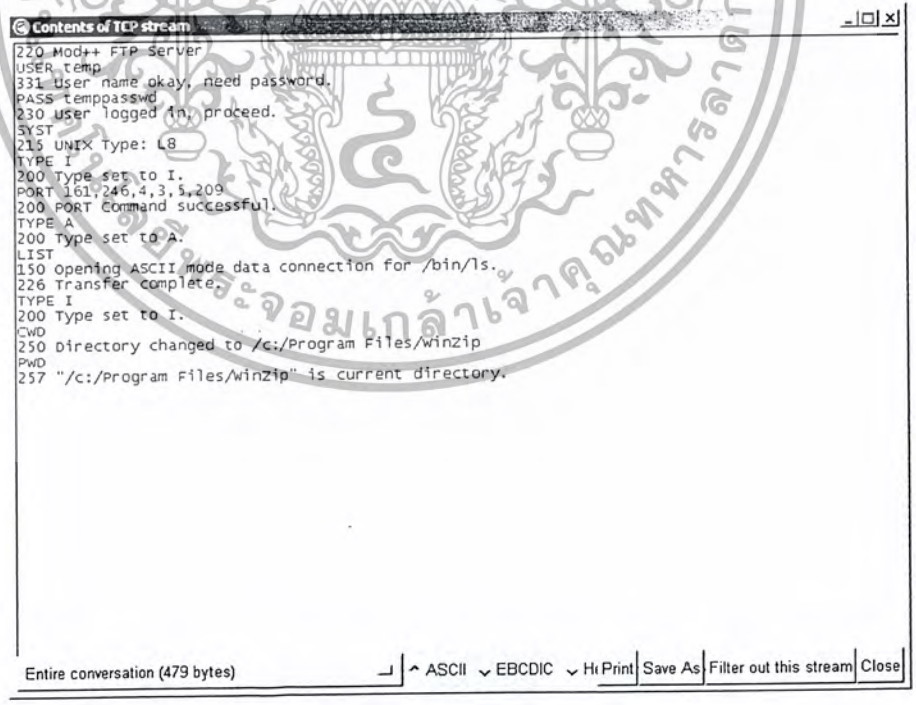
ผลลัพธ์ที่ได้จากการดักจับ

ผลลัพธ์ที่ได้จากเงื่อนไข "TCP PORT 20 OR TCP PORT 21" เป็นดังรูป ข-04 การแสดงผลจะประกอบด้วย 3 ส่วนคือ ส่วนที่แสดงแพ็กเก็ตทั้งหมดเรียงตามลำดับที่ได้รับ ส่วนที่แสดงข้อมูลที่ถอดรหัส (decode) ตามแต่ละโพรโตคอลหรือมาตรฐาน และส่วนสุดท้ายคือ raw ethernet frame ที่ส่งกันในเครือข่าย



รูป ข-04 ตัวอย่างผลลัพธ์จากการใช้เงื่อนไขกรองแพ็กเก็ต

เมื่อคลิ๊กขวาแล้วเลือก Follow TCP Stream จะสามารถถอดรหัสออกมาเป็นข้อมูลที่ต่อเนื่องกันในแต่ละการเชื่อมต่อได้ดังรูป ข-05



รูป ข-05 Follow TCP Stream

ภาคผนวก ค

Network Virtual Terminal

Communication is established using the TCP/IP protocols and communication is based on a set of facilities known as a **Network Virtual Terminal** (NVT). At the user or client end the telnet client program is responsible for mapping incoming NVT codes to the actual codes needed to operate the user's display device and is also responsible for mapping user generated keyboard sequences into NVT sequences.

The NVT uses 7 bit codes for characters, the display device, referred to as a printer in the RFC, is only required to display the "standard" printing ASCII characters represented by 7 bit codes and to recognise and process certain control codes. The 7 bit characters are transmitted as 8 bit bytes with most significant bit set to zero. An end-of-line is transmitted as the character sequence CR (carriage return) followed by LF (line feed). If it is desired to transmit an actual carriage return this is transmitted as a carriage return followed by a NUL (all bits zero) character.

NVT ASCII is used by many other Internet protocols.

The following control codes are required to be understood by the Network Virtual Terminal.

Name	code	Decimal Value	Function
NULL	NUL	0	No operation
Line Feed	LF	10	Moves the printer to the next print line, keeping the same horizontal position.
Carriage Return	CR	13	Moves the printer to the left margin of the current line.
BELL	BEL	7	Produces an audible or visible signal (which does NOT move the print head.
Back Space	BS	8	Moves the print head one character position towards the left margin. [On a printing devices this mechanism was commonly used to form composite characters by printing two basic characters on top of each other.]

Horizontal Tab	HT	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Vertical Tab	VT	11	Moves the printer to the next vertical tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Form Feed	FF	12	Moves the printer to the top of the next page, keeping the same horizontal position. [On visual displays this commonly clears the screen and moves the cursor to the top left corner.]

The NVT keyboard is specified as being capable of generating all 128 ASCII codes by using keys, key combinations or key sequences.

ภาคผนวก ง

ANSI/VT100 Terminal Control

Many computer terminals and terminal emulators support color and cursor control through a system of escape sequences. One such standard is commonly referred to as ANSI Color. Several terminal specifications are based on the ANSI color standard, including VT100.

The following is a partial listing of the VT100 control set.

<ESC> represents the ANSI "escape" character, 0x1B. Bracketed tags represent modifiable decimal parameters; eg. {ROW} would be replaced by a row number.

Device Status

The following codes are used for reporting terminal/display settings, and vary depending on the implementation:

Query Device Code **<ESC>[c**

Requests a **Report Device Code** response from the device.

Report Device Code **<ESC>[{code}0c**

Generated by the device in response to **Query Device Code** request.

Query Device Status **<ESC>[5n**

Requests a **Report Device Status** response from the device.

Report Device OK **<ESC>[0n**

Generated by the device in response to a **Query Device Status** request; indicates that device is functioning correctly.

Report Device Failure **<ESC>[3n**

Generated by the device in response to a **Query Device Status** request; indicates that device is functioning improperly.

Query Cursor Position **<ESC>[6n**

Requests a **Report Cursor Position** response from the device.

Report Cursor Position **<ESC>[{ROW};{COLUMN}R**

Generated by the device in response to a **Query Cursor Position** request; reports current cursor position.

Terminal Setup

The **h** and **l** codes are used for setting terminal/display mode, and vary depending on the implementation. Line Wrap is one of the few setup codes that tend to be used consistently:

Reset Device **<ESC>c**

Reset all terminal settings to default.

Enable Line Wrap **<ESC>[7h**

Text wraps to next line if longer than the length of the display area.

Disable Line Wrap <ESC>[7l

Disables line wrapping.

Fonts

Some terminals support multiple fonts: normal/bold, swiss/italic, etc. There are a variety of special codes for certain terminals; the following are fairly standard:

Font Set G0 **<ESC>(**

Set default font.

Font Set G1 **<ESC>)**

Set alternate font.

Cursor Control

Cursor Home **<ESC>[{ROW};{COLUMN}H**

Sets the cursor position where subsequent text will begin. If no row/column parameters are provided (ie. <ESC>[H), the cursor will move to the *home* position, at the upper left of the screen.

Cursor Up <ESC>[{COUNT}A

Moves the cursor up by *COUNT* rows; the default count is 1.

Cursor Down **<ESC>[{COUNT}B**

Moves the cursor down by *COUNT* rows; the default count is 1.

Cursor Forward **<ESC>[{COUNT}C**

Moves the cursor *forward* by *COUNT* columns; the default count is 1.

Force Cursor Position <ESC>[{ROW};{COLUMN}f

Identical to **Cursor Home**.

Save Cursor **<ESC>[s**

Save current cursor position.

Unsave Cursor **<ESC>[u**

Restores cursor position after a **Save Cursor**.

Save Cursor & Attrs **<ESC>7**

Save current cursor position.

Restore Cursor & Attrs **<ESC>8**

Restores cursor position after a **Save Cursor**.

Scrolling

Scroll Screen **<ESC>[r**

Enable scrolling for entire display.

Scroll Screen **<ESC>[{start};{end}r**

Enable scrolling from row {start} to row {end}.

Scroll Down **<ESC>D**

Scroll display down one line.

Scroll Up **<ESC>M**

Scroll display up one line.

Tab Control

Set Tab **<ESC>H**

Sets a tab at the current position.

Clear Tab **<ESC>[g**

Clears tab at the current position.

Clear All Tabs **<ESC>[3g**

Clears all tabs.

Erasing Text

Erase End of Line **<ESC>[K**

Erases from the current cursor position to the end of the current line.

Erase Start of Line **<ESC>[1K**

Erases from the current cursor position to the start of the current line.

Erase Line **<ESC>[2K**

Erases the entire current line.

Erase Down **<ESC>[J**

Erases the screen from the current line down to the bottom of the screen.

Erase Up **<ESC>[1J**

Erases the screen from the current line up to the top of the screen.

Erase Screen **<ESC>[2J**

Erases the screen with the background color and moves the cursor to *home*.

Printing

Some terminals support local printing:

Print Screen **<ESC>[i**

Print the current screen.

Print Line **<ESC>[1i**

Print the current line.

Stop Print Log **<ESC>[4i**

Disable log.

Start Print Log **<ESC>[5i**

Start log; all received text is echoed to a printer.

Define Key

Set Key Definition **<ESC>[{key};"{string}"p**

Associates a *string* of text to a keyboard key. {key} indicates the key by its ASCII value in decimal.

Set Display Attributes

Set Attribute Mode **<ESC>[{attr1};...;{attrn}m**

Sets multiple display attribute settings. The following lists standard attributes:

0 Reset all attributes

1 Bright

2 Dim

4 Underscore

5 Blink

7	Reverse
8	Hidden

Foreground Colors

30	Black
31	Red
32	Green
33	Yellow
34	Blue
35	Magenta
36	Cyan
37	White

Background Colors

40	Black
41	Red
42	Green
43	Yellow
44	Blue
45	Magenta
46	Cyan
47	White

ภาคผนวก จ

SSH Protocol Architecture (SSH-ARCH)

Network Working Group
Internet-Draft
Expires: March 31, 2004

T. Ylonen
SSH Communications Security Corp
D. Moffat, Ed.
Sun Microsystems, Inc
Oct 2003

SSH Protocol Architecture
draft-ietf-secsh-architecture-15.txt

Status of this Memo

Copyright Notice

Abstract

SSH is a protocol for secure remote login and other secure network services over an insecure network. This document describes the architecture of the SSH protocol, as well as the notation and terminology used in SSH protocol documents. It also discusses the SSH algorithm naming system that allows local extensions. The SSH protocol consists of three major components: The Transport Layer Protocol provides server authentication, confidentiality, and integrity with perfect forward secrecy. The User Authentication Protocol authenticates the client to the server. The Connection Protocol multiplexes the encrypted tunnel into several logical channels. Details of these protocols are described in separate

documents.

Table of Contents

1. Contributors 3
2. Introduction 3
3. Specification of Requirements 3
4. Architecture 3
4.1 Host Keys 4
4.2 Extensibility 5
4.3 Policy Issues 5
4.4 Security Properties 6
4.5 Packet Size and Overhead 6
4.6 Localization and Character Set Support 7
5. Data Type Representations Used in the SSH Protocols 8
6. Algorithm Naming 10
7. Message Numbers 11
8. IANA Considerations 11
9. Security Considerations 12
9.1 Pseudo-Random Number Generation 12
9.2 Transport 13
9.2.1 Confidentiality 13
9.2.2 Data Integrity 16
9.2.3 Replay 16
9.2.4 Man-in-the-middle 17
9.2.5 Denial-of-service 19
9.2.6 Covert Channels 19
9.2.7 Forward Secrecy 20
9.3 Authentication Protocol 20
9.3.1 Weak Transport 21
9.3.2 Debug messages 21
9.3.3 Local security policy 21
9.3.4 Public key authentication 22
9.3.5 Password authentication 22
9.3.6 Host based authentication 23
9.4 Connection protocol 23
9.4.1 End point security 23
9.4.2 Proxy forwarding 23
9.4.3 X11 forwarding 24
Normative References 24
Informative References 25
Authors' Addresses 27
Intellectual Property and Copyright Statements 28

1. Contributors

The major original contributors of this document were: Tatu Ylonen, Tero Kivinen, Timo J. Rinne, Sami Lehtinen (all of SSH Communications Security Corp), and Markku-Juhani O. Saarinen (University of Jyvaskyla)

The document editor is: Darren.Moffat@Sun.COM. Comments on this internet draft should be sent to the IETF SECSH working group, details at: http://ietf.org/html.charters/secsh-charter.html

2. Introduction

SSH is a protocol for secure remote login and other secure network services over an insecure network. It consists of three major components:

- The Transport Layer Protocol [SSH-TRANS] provides server authentication, confidentiality, and integrity. It may optionally also provide compression. The transport layer will typically be run over a TCP/IP connection, but might also be used on top of any other reliable data stream.
- The User Authentication Protocol [SSH-USERAUTH] authenticates the client-side user to the server. It runs over the transport layer protocol.
- The Connection Protocol [SSH-CONNECT] multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.

The client sends a service request once a secure transport layer connection has been established. A second service request is sent after user authentication is complete. This allows new protocols to be defined and coexist with the protocols listed above.

The connection protocol provides channels that can be used for a wide range of purposes. Standard methods are provided for setting up secure interactive shell sessions and for forwarding ("tunneling") arbitrary TCP/IP ports and X11 connections.

3. Specification of Requirements

All documents related to the SSH protocols shall use the keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" to describe requirements. They are to be interpreted as described in [RFC2119].

4. Architecture

4.1 Host Keys

Each server host SHOULD have a host key. Hosts MAY have multiple host keys using multiple different algorithms. Multiple hosts MAY share the same host key. If a host has keys at all, it MUST have at least one key using each REQUIRED public key algorithm (DSS [FIPS-186]).

The server host key is used during key exchange to verify that the client is really talking to the correct server. For this to be possible, the client must have a priori knowledge of the server's public host key.

Two different trust models can be used:

- The client has a local database that associates each host name (as typed by the user) with the corresponding public host key. This method requires no centrally administered infrastructure, and no third-party coordination. The downside is that the database of name-to-key associations may become burdensome to maintain.
- The host name-to-key association is certified by some trusted certification authority. The client only knows the CA root key, and can verify the validity of all host keys certified by accepted CAs.

The second alternative eases the maintenance problem, since ideally only a single CA key needs to be securely stored on the client. On the other hand, each host key must be appropriately certified by a central authority before authorization is possible. Also, a lot of trust is placed on the central infrastructure.

The protocol provides the option that the server name - host key association is not checked when connecting to the host for the first time. This allows communication without prior communication of host keys or certification. The connection still provides protection against passive listening; however, it becomes vulnerable to active man-in-the-middle attacks. Implementations SHOULD NOT normally allow such connections by default, as they pose a potential security problem. However, as there is no widely deployed key infrastructure available on the Internet yet, this option makes the protocol much more usable during the transition time until such an infrastructure emerges, while still providing a much higher level of security than that offered by older solutions (e.g. telnet [RFC-854] and rlogin [RFC-1282]).

Implementations SHOULD try to make the best effort to check host keys. An example of a possible strategy is to only accept a host key without checking the first time a host is connected, save the key in a local database, and compare against that key on all future

connections to that host.

Implementations MAY provide additional methods for verifying the correctness of host keys, e.g. a hexadecimal fingerprint derived from the SHA-1 hash of the public key. Such fingerprints can easily be verified by using telephone or other external communication channels.

All implementations SHOULD provide an option to not accept host keys that cannot be verified.

We believe that ease of use is critical to end-user acceptance of security solutions, and no improvement in security is gained if the new solutions are not used. Thus, providing the option not to check the server host key is believed to improve the overall security of the Internet, even though it reduces the security of the protocol in configurations where it is allowed.

4.2 Extensibility

We believe that the protocol will evolve over time, and some organizations will want to use their own encryption, authentication and/or key exchange methods. Central registration of all extensions is cumbersome, especially for experimental or classified features. On the other hand, having no central registration leads to conflicts in method identifiers, making interoperability difficult.

We have chosen to identify algorithms, methods, formats, and extension protocols with textual names that are of a specific format. DNS names are used to create local namespaces where experimental or classified extensions can be defined without fear of conflicts with other implementations.

One design goal has been to keep the base protocol as simple as possible, and to require as few algorithms as possible. However, all implementations MUST support a minimal set of algorithms to ensure interoperability (this does not imply that the local policy on all hosts would necessary allow these algorithms). The mandatory algorithms are specified in the relevant protocol documents.

Additional algorithms, methods, formats, and extension protocols can be defined in separate drafts. See Section Algorithm Naming (Section 6) for more information.

4.3 Policy Issues

The protocol allows full negotiation of encryption, integrity, key exchange, compression, and public key algorithms and formats. Encryption, integrity, public key, and compression algorithms can be

different for each direction.

The following policy issues SHOULD be addressed in the configuration mechanisms of each implementation:

- Encryption, integrity, and compression algorithms, separately for each direction. The policy MUST specify which is the preferred algorithm (e.g. the first algorithm listed in each category).
- Public key algorithms and key exchange method to be used for host authentication. The existence of trusted host keys for different public key algorithms also affects this choice.
- The authentication methods that are to be required by the server for each user. The server's policy MAY require multiple authentication for some or all users. The required algorithms MAY depend on the location where the user is trying to log in from.
- The operations that the user is allowed to perform using the connection protocol. Some issues are related to security; for example, the policy SHOULD NOT allow the server to start sessions or run commands on the client machine, and MUST NOT allow connections to the authentication agent unless forwarding such connections has been requested. Other issues, such as which TCP/IP ports can be forwarded and by whom, are clearly issues of local policy. Many of these issues may involve traversing or bypassing firewalls, and are interrelated with the local security policy.

4.4 Security Properties

The primary goal of the SSH protocol is improved security on the Internet. It attempts to do this in a way that is easy to deploy, even at the cost of absolute security.

- All encryption, integrity, and public key algorithms used are well-known, well-established algorithms.
- All algorithms are used with cryptographically sound key sizes that are believed to provide protection against even the strongest cryptanalytic attacks for decades.
- All algorithms are negotiated, and in case some algorithm is broken, it is easy to switch to some other algorithm without modifying the base protocol.

Specific concessions were made to make wide-spread fast deployment easier. The particular case where this comes up is verifying that the server host key really belongs to the desired host; the protocol allows the verification to be left out (but this is NOT RECOMMENDED). This is believed to significantly improve usability in the short term, until widespread Internet public key infrastructures emerge.

4.5 Packet Size and Overhead

Some readers will worry about the increase in packet size due to new

headers, padding, and MAC. The minimum packet size is in the order of 28 bytes (depending on negotiated algorithms). The increase is negligible for large packets, but very significant for one-byte packets (telnet-type sessions). There are, however, several factors that make this a non-issue in almost all cases:

- The minimum size of a TCP/IP header is 32 bytes. Thus, the increase is actually from 33 to 51 bytes (roughly).
- The minimum size of the data field of an Ethernet packet is 46 bytes [RFC-894]. Thus, the increase is no more than 5 bytes. When Ethernet headers are considered, the increase is less than 10 percent.
- The total fraction of telnet-type data in the Internet is negligible, even with increased packet sizes.

The only environment where the packet size increase is likely to have a significant effect is PPP [RFC-1134] over slow modem lines (PPP compresses the TCP/IP headers, emphasizing the increase in packet size). However, with modern modems, the time needed to transfer is in the order of 2 milliseconds, which is a lot faster than people can type.

There are also issues related to the maximum packet size. To minimize delays in screen updates, one does not want excessively large packets for interactive sessions. The maximum packet size is negotiated separately for each channel.

4.6 Localization and Character Set Support

For the most part, the SSH protocols do not directly pass text that would be displayed to the user. However, there are some places where such data might be passed. When applicable, the character set for the data MUST be explicitly specified. In most places, ISO 10646 with UTF-8 encoding is used [RFC-2279]. When applicable, a field is also provided for a language tag [RFC-3066].

One big issue is the character set of the interactive session. There is no clear solution, as different applications may display data in different formats. Different types of terminal emulation may also be employed in the client, and the character set to be used is effectively determined by the terminal emulation. Thus, no place is provided for directly specifying the character set or encoding for terminal session data. However, the terminal emulation type (e.g. "vt100") is transmitted to the remote site, and it implicitly specifies the character set and encoding. Applications typically use the terminal type to determine what character set they use, or the character set is determined using some external means. The terminal emulation may also allow configuring the default character set. In any case, the character set for the terminal session is considered

primarily a client local issue.

Internal names used to identify algorithms or protocols are normally never displayed to users, and must be in US-ASCII.

The client and server user names are inherently constrained by what the server is prepared to accept. They might, however, occasionally be displayed in logs, reports, etc. They MUST be encoded using ISO 10646 UTF-8, but other encodings may be required in some cases. It is up to the server to decide how to map user names to accepted user names. Straight bit-wise binary comparison is RECOMMENDED.

For localization purposes, the protocol attempts to minimize the number of textual messages transmitted. When present, such messages typically relate to errors, debugging information, or some externally configured data. For data that is normally displayed, it SHOULD be possible to fetch a localized message instead of the transmitted message by using a numerical code. The remaining messages SHOULD be configurable.

5. Data Type Representations Used in the SSH Protocols

byte

A byte represents an arbitrary 8-bit value (octet) [RFC-1700]. Fixed length data is sometimes represented as an array of bytes, written byte[n], where n is the number of bytes in the array.

boolean

A boolean value is stored as a single byte. The value 0 represents FALSE, and the value 1 represents TRUE. All non-zero values MUST be interpreted as TRUE; however, applications MUST NOT store values other than 0 and 1.

uint32

Represents a 32-bit unsigned integer. Stored as four bytes in the order of decreasing significance (network byte order). For example, the value 699921578 (0x29b7f4aa) is stored as 29 b7 f4 aa.

uint64

Represents a 64-bit unsigned integer. Stored as eight bytes in the order of decreasing significance (network byte order).

string

Arbitrary length binary string. Strings are allowed to contain arbitrary binary data, including null characters and 8-bit characters. They are stored as a uint32 containing its length (number of bytes that follow) and zero (= empty string) or more bytes that are the value of the string. Terminating null characters are not used.

Strings are also used to store text. In that case, US-ASCII is used for internal names, and ISO-10646 UTF-8 for text that might be displayed to the user. The terminating null character SHOULD NOT normally be stored in the string.

For example, the US-ASCII string "testing" is represented as 00 00 00 07 t e s t i n g. The UTF8 mapping does not alter the encoding of US-ASCII characters.

mpint

Represents multiple precision integers in two's complement format, stored as a string, 8 bits per byte, MSB first. Negative numbers have the value 1 as the most significant bit of the first byte of the data partition. If the most significant bit would be set for a positive number, the number MUST be preceded by a zero byte. Unnecessary leading bytes with the value 0 or 255 MUST NOT be included. The value zero MUST be stored as a string with zero bytes of data.

By convention, a number that is used in modular computations in Z_n SHOULD be represented in the range 0 <= x < n.

Examples:

value (hex)	representation (hex)
0	00 00 00 00
9a378f9b2e332a7	00 00 00 08 09 a3 78 f9 b2 e3 32 a7
80	00 00 00 02 00 80
-1234	00 00 00 02 ed cc
-deadbeef	00 00 00 05 ff 21 52 41 11

name-list

A string containing a comma separated list of names. A name list is represented as a uint32 containing its length (number of bytes that follow) followed by a comma-separated list of zero or more

names. A name MUST be non-zero length, and it MUST NOT contain a comma (','). Context may impose additional restrictions on the names; for example, the names in a list may have to be valid algorithm identifier (see Algorithm Naming below), or [RFC-3066] language tags. The order of the names in a list may or may not be significant, also depending on the context where the list is is used. Terminating NUL characters are not used, neither for the individual names, nor for the list as a whole.

Examples:

value	representation (hex)
(), the empty list	00 00 00 00
("zlib")	00 00 00 04 7a 6c 69 62
("zlib", "none")	00 00 00 09 7a 6c 69 62 2c 6e 6f 6e 65

6. Algorithm Naming

The SSH protocols refer to particular hash, encryption, integrity, compression, and key exchange algorithms or protocols by names. There are some standard algorithms that all implementations MUST support. There are also algorithms that are defined in the protocol specification but are OPTIONAL. Furthermore, it is expected that some organizations will want to use their own algorithms.

In this protocol, all algorithm identifiers MUST be printable US-ASCII non-empty strings no longer than 64 characters. Names MUST be case-sensitive.

There are two formats for algorithm names:

- Names that do not contain an at-sign (@) are reserved to be assigned by IETF consensus (RFCs). Examples include `3des-cbc', `sha-1', `hmac-sha1', and `zlib' (the quotes are not part of the name). Names of this format MUST NOT be used without first registering them. Registered names MUST NOT contain an at-sign (@) or a comma (,).
- Anyone can define additional algorithms by using names in the format name@domainname, e.g. "ourcipher-cbc@example.com". The format of the part preceding the at sign is not specified; it MUST consist of US-ASCII characters except at-sign and comma. The part following the at-sign MUST be a valid fully qualified internet domain name [RFC-1034] controlled by the person or organization defining the name. It is up to each domain how it manages its local namespace.

7. Message Numbers

SSH packets have message numbers in the range 1 to 255. These numbers have been allocated as follows:

Transport layer protocol:

1 to 19	Transport layer generic (e.g. disconnect, ignore, debug, etc.)
20 to 29	Algorithm negotiation
30 to 49	Key exchange method specific (numbers can be reused for different authentication methods)

User authentication protocol:

50 to 59	User authentication generic
60 to 79	User authentication method specific (numbers can be reused for different authentication methods)

Connection protocol:

80 to 89	Connection protocol generic
90 to 127	Channel related messages

Reserved for client protocols:

128 to 191	Reserved

Local extensions:

192 to 255	Local extensions

8. IANA Considerations

The initial state of the IANA registry is detailed in [SSH-NUMBERS].

Allocation of the following types of names in the SSH protocols is assigned by IETF consensus:

- SSH encryption algorithm names,
- SSH MAC algorithm names,
- SSH public key algorithm names (public key algorithm also implies encoding and signature/encryption capability),
- SSH key exchange method names, and
- SSH protocol (service) names.

These names MUST be printable US-ASCII strings, and MUST NOT contain the characters at-sign ('@'), comma (','), or whitespace or control characters (ASCII codes 32 or less). Names are case-sensitive, and MUST NOT be longer than 64 characters.

Names with the at-sign ('@') in them are allocated by the owner of DNS name after the at-sign (hierarchical allocation in [RFC-2343]), otherwise the same restrictions as above.

Each category of names listed above has a separate namespace. However, using the same name in multiple categories SHOULD be avoided to minimize confusion.

Message numbers (see Section Message Numbers (Section 7)) in the range of 0..191 are allocated via IETF consensus; message numbers in the 192..255 range (the "Local extensions" set) are reserved for private use.

9. Security Considerations

In order to make the entire body of Security Considerations more accessible, Security Considerations for the transport, authentication, and connection documents have been gathered here.

The transport protocol [1] provides a confidential channel over an insecure network. It performs server host authentication, key exchange, encryption, and integrity protection. It also derives a unique session id that may be used by higher-level protocols.

The authentication protocol [2] provides a suite of mechanisms which can be used to authenticate the client user to the server. Individual mechanisms specified in the in authentication protocol use the session id provided by the transport protocol and/or depend on the security and integrity guarantees of the transport protocol.

The connection protocol [3] specifies a mechanism to multiplex multiple streams [channels] of data over the confidential and authenticated transport. It also specifies channels for accessing an interactive shell, for 'proxy-forwarding' various external protocols over the secure transport (including arbitrary TCP/IP protocols), and for accessing secure 'subsystems' on the server host.

9.1 Pseudo-Random Number Generation

This protocol binds each session key to the session by including random, session specific data in the hash used to produce session keys. Special care should be taken to ensure that all of the random numbers are of good quality. If the random data here (e.g., DH

parameters) are pseudo-random then the pseudo-random number generator should be cryptographically secure (i.e., its next output not easily guessed even when knowing all previous outputs) and, furthermore, proper entropy needs to be added to the pseudo-random number generator. RFC 1750 [1750] offers suggestions for sources of random numbers and entropy. Implementors should note the importance of entropy and the well-meant, anecdotal warning about the difficulty in properly implementing pseudo-random number generating functions.

The amount of entropy available to a given client or server may sometimes be less than what is required. In this case one must either resort to pseudo-random number generation regardless of insufficient entropy or refuse to run the protocol. The latter is preferable.

9.2 Transport

9.2.1 Confidentiality

It is beyond the scope of this document and the Secure Shell Working Group to analyze or recommend specific ciphers other than the ones which have been established and accepted within the industry. At the time of this writing, ciphers commonly in use include 3DES, ARCFOUR, twofish, serpent and blowfish. AES has been accepted by The published as a US Federal Information Processing Standards [FIPS-197] and the cryptographic community as being acceptable for this purpose as well has accepted AES. As always, implementors and users should check current literature to ensure that no recent vulnerabilities have been found in ciphers used within products. Implementors should also check to see which ciphers are considered to be relatively stronger than others and should recommend their use to users over relatively weaker ciphers. It would be considered good form for an implementation to politely and unobtrusively notify a user that a stronger cipher is available and should be used when a weaker one is actively chosen.

The "none" cipher is provided for debugging and SHOULD NOT be used except for that purpose. It's cryptographic properties are sufficiently described in RFC 2410, which will show that its use does not meet the intent of this protocol.

The relative merits of these and other ciphers may also be found in current literature. Two references that may provide information on the subject are [SCHNEIER] and [KAUFMAN,PERLMAN,SPECINER]. Both of these describe the CBC mode of operation of certain ciphers and the weakness of this scheme. Essentially, this mode is theoretically vulnerable to chosen cipher-text attacks because of the high predictability of the start of packet sequence. However, this attack

is still deemed difficult and not considered fully practicable especially if relatively longer block sizes are used.

Additionally, another CBC mode attack may be mitigated through the insertion of packets containing SSH_MSG_IGNORE. Without this technique, a specific attack may be successful. For this attack (commonly known as the Rogaway attack [ROGAWAY],[DAI],[BELLARE,KOHNO,NAMPREMPRE]) to work, the attacker would need to know the IV of the next block that is going to be encrypted. In CBC mode that is the output of the encryption of the previous block. If the attacker does not have any way to see the packet yet (i.e it is in the internal buffers of the ssh implementation or even in the kernel) then this attack will not work. If the last packet has been sent out to the network (i.e the attacker has access to it) then he can use the attack.

In the optimal case an implementor would need to add an extra packet only if the packet has been sent out onto the network and there are no other packets waiting for transmission. Implementors may wish to check to see if there are any unsent packets awaiting transmission, but unfortunately it is not normally easy to obtain this information from the kernel or buffers. If there are not, then a packet containing SSH_MSG_IGNORE SHOULD be sent. If a new packet is added to the stream every time the attacker knows the IV that is supposed to be used for the next packet, then the attacker will not be able to guess the correct IV, thus the attack will never be successfull.

As an example, consider the following case:

```
   Client                                              Server
   ------                                              ------
   TCP(seq=x, len=500)          ->
 contains Record 1

                          [500 ms passes, no ACK]

TCP(seq=x, len=1000)          ->
 contains Records 1,2

                                                        ACK
```

1. The Nagle algorithm + TCP retransmits mean that the two records get coalesced into a single TCP segment
2. Record 2 is *not* at the beginning of the TCP segment and never will be, since it gets ACKed.

3. Yet, the attack is possible because Record 1 has already been seen.

As this example indicates, it's totally unsafe to use the existence of unflushed data in the TCP buffers proper as a guide to whether you need an empty packet, since when you do the second write(), the buffers will contain the un-ACKed Record 1.

On the other hand, it's perfectly safe to have the following situation:

```
Client                                                  Server
------                                                  ------
TCP(seq=x, len=500)                 ->
   contains SSH_MSG_IGNORE

TCP(seq=y, len=500)                 ->
   contains Data
```

Provided that the IV for second SSH Record is fixed after the data for the Data packet is determined -i.e. you do:

```
read from user
encrypt null packet
encrypt data packet
```

9.2.2 Data Integrity

This protocol does allow the Data Integrity mechanism to be disabled. Implementors SHOULD be wary of exposing this feature for any purpose other than debugging. Users and administrators SHOULD be explicitly warned anytime the "none" MAC is enabled.

So long as the "none" MAC is not used, this protocol provides data integrity.

Because MACs use a 32 bit sequence number, they might start to leak information after 2**32 packets have been sent. However, following the rekeying recommendations should prevent this attack. The transport protocol [1] recommends rekeying after one gigabyte of data, and the smallest possible packet is 16 bytes. Therefore, rekeying SHOULD happen after 2**28 packets at the very most.

9.2.3 Replay

The use of a MAC other than 'none' provides integrity and authentication. In addition, the transport protocol provides a unique session identifier (bound in part to pseudo-random data that is part of the algorithm and key exchange process) that can be used by higher level protocols to bind data to a given session and prevent replay of data from prior sessions. For example, the authentication protocol uses this to prevent replay of signatures from previous sessions. Because public key authentication exchanges are cryptographically bound to the session (i.e., to the initial key exchange) they cannot be successfully replayed in other sessions.

Note that the session ID can be made public without harming the security of the protocol.

If two session happen to have the same session ID [hash of key exchanges] then packets from one can be replayed against the other. It must be stressed that the chances of such an occurrence are, needless to say, minimal when using modern cryptographic methods. This is all the more so true when specifying larger hash function outputs and DH parameters.

Replay detection using monotonically increasing sequence numbers as input to the MAC, or HMAC in some cases, is described in [RFC2085] /> [RFC2246], [RFC2743], [RFC1964], [RFC2025], and [RFC1510]. The underlying construct is discussed in [RFC2104]. Essentially a different sequence number in each packet ensures that at least this one input to the MAC function will be unique and will provide a nonrecurring MAC output that is not predictable to an attacker. If the session stays active long enough, however, this sequence number will wrap. This event may provide an attacker an opportunity to replay a previously recorded packet with an identical sequence number but only if the peers have not rekeyed since the transmission of the first packet with that sequence number. If the peers have rekeyed, then the replay will be detected as the MAC check will fail. For this reason, it must be emphasized that peers MUST rekey before a wrap of the sequence numbers. Naturally, if an attacker does attempt to replay a captured packet before the peers have rekeyed, then the receiver of the duplicate packet will not be able to validate the MAC and it will be discarded. The reason that the MAC will fail is because the receiver will formulate a MAC based upon the packet contents, the shared secret, and the expected sequence number. Since the replayed packet will not be using that expected sequence number (the sequence number of the replayed packet will have already been passed by the receiver) then the calculated MAC will not match the MAC received with the packet.

9.2.4 Man-in-the-middle

This protocol makes no assumptions nor provisions for an infrastructure or means for distributing the public keys of hosts. It is expected that this protocol will sometimes be used without first verifying the association between the server host key and the server host name. Such usage is vulnerable to man-in-the-middle attacks. This section describes this and encourages administrators and users to understand the importance of verifying this association before any session is initiated.

There are three cases of man-in-the-middle attacks to consider. The first is where an attacker places a device between the client and the

server before the session is initiated. In this case, the attack device is trying to mimic the legitimate server and will offer its public key to the client when the client initiates a session. If it were to offer the public key of the server, then it would not be able to decrypt or sign the transmissions between the legitimate server and the client unless it also had access to the private-key of the host. The attack device will also, simultaneously to this, initiate a session to the legitimate server masquerading itself as the client. If the public key of the server had been securely distributed to the client prior to that session initiation, the key offered to the client by the attack device will not match the key stored on the client. In that case, the user SHOULD be given a warning that the offered host key does not match the host key cached on the client. As described in Section 3.1 of [ARCH], the user may be free to accept the new key and continue the session. It is RECOMMENDED that the warning provide sufficient information to the user of the client device so they may make an informed decision. If the user chooses to continue the session with the stored public-key of the server (not the public-key offered at the start of the session), then the session specific data between the attacker and server will be different between the client-to-attacker session and the attacker-to-server sessions due to the randomness discussed above. From this, the attacker will not be able to make this attack work since the attacker will not be able to correctly sign packets containing this session specific data from the server since he does not have the private key of that server.

The second case that should be considered is similar to the first case in that it also happens at the time of connection but this case points out the need for the secure distribution of server public keys. If the server public keys are not securely distributed then the client cannot know if it is talking to the intended server. An attacker may use social engineering techniques to pass off server keys to unsuspecting users and may then place a man-in-the-middle attack device between the legitimate server and the clients. If this is allowed to happen then the clients will form client-to-attacker sessions and the attacker will form attacker-to-server sessions and will be able to monitor and manipulate all of the traffic between the clients and the legitimate servers. Server administrators are encouraged to make host key fingerprints available for checking by some means whose security does not rely on the integrity of the actual host keys. Possible mechanisms are discussed in Section 3.1 of [SSH-ARCH] and may also include secured Web pages, physical pieces of paper, etc. Implementors SHOULD provide recommendations on how best to do this with their implementation. Because the protocol is extensible, future extensions to the protocol may provide better mechanisms for dealing with the need to know the server's host key before connecting. For example, making the host key fingerprint

available through a secure DNS lookup, or using kerberos over gssapi during key exchange to authenticate the server are possibilities.

In the third man-in-the-middle case, attackers may attempt to manipulate packets in transit between peers after the session has been established. As described in the Replay part of this section, a successful attack of this nature is very improbable. As in the Replay section, this reasoning does assume that the MAC is secure and that it is infeasible to construct inputs to a MAC algorithm to give a known output. This is discussed in much greater detail in Section 6 of RFC 2104. If the MAC algorithm has a vulnerability or is weak enough, then the attacker may be able to specify certain inputs to yield a known MAC. With that they may be able to alter the contents of a packet in transit. Alternatively the attacker may be able to exploit the algorithm vulnerability or weakness to find the shared secret by reviewing the MACs from captured packets. In either of those cases, an attacker could construct a packet or packets that could be inserted into an SSH stream. To prevent that, implementors are encouraged to utilize commonly accepted MAC algorithms and administrators are encouraged to watch current literature and discussions of cryptography to ensure that they are not using a MAC algorithm that has a recently found vulnerability or weakness.

In summary, the use of this protocol without a reliable association of the binding between a host and its host keys is inherently insecure and is NOT RECOMMENDED. It may however be necessary in non-security critical environments, and will still provide protection against passive attacks. Implementors of protocols and applications running on top of this protocol should keep this possibility in mind.

9.2.5 Denial-of-service

This protocol is designed to be used over a reliable transport. If transmission errors or message manipulation occur, the connection is closed. The connection SHOULD be re-established if this occurs. Denial of service attacks of this type ("wire cutter") are almost impossible to avoid.

In addition, this protocol is vulnerable to Denial of Service attacks because an attacker can force the server to go through the CPU and memory intensive tasks of connection setup and key exchange without authenticating. Implementors SHOULD provide features that make this more difficult. For example, only allowing connections from a subset of IPs known to have valid users.

9.2.6 Covert Channels

The protocol was not designed to eliminate covert channels. For

example, the padding, SSH_MSG_IGNORE messages, and several other places in the protocol can be used to pass covert information, and the recipient has no reliable way to verify whether such information is being sent.

- 9.2.7 Forward Secrecy

It should be noted that the Diffie-Hellman key exchanges may provide perfect forward secrecy (PFS). PFS is essentially defined as the cryptographic property of a key-establishment protocol in which the compromise of a session key or long-term private key after a given session does not cause the compromise of any earlier session. [ANSI T1.523-2001] SSHv2 sessions resulting from a key exchange using diffie-hellman-group1-sha1 are secure even if private keying/ authentication material is later revealed, but not if the session keys are revealed. So, given this definition of PFS, SSHv2 does have PFS. It is hoped that all other key exchange mechanisms proposed and used in the future will also provide PFS. This property is not commuted to any of the applications or protocols using SSH as a transport however. The transport layer of SSH provides confidentiality for password authentication and other methods that rely on secret data.

Of course, if the DH private parameters for the client and server are revealed then the session key is revealed, but these items can be thrown away after the key exchange completes. It's worth pointing out that these items should not be allowed to end up on swap space and that they should be erased from memory as soon as the key exchange completes.

9.3 Authentication Protocol

The purpose of this protocol is to perform client user authentication. It assumes that this run over a secure transport layer protocol, which has already authenticated the server machine, established an encrypted communications channel, and computed a unique session identifier for this session.

Several authentication methods with different security characteristics are allowed. It is up to the server's local policy to decide which methods (or combinations of methods) it is willing to accept for each user. Authentication is no stronger than the weakest combination allowed.

The server may go into a "sleep" period after repeated unsuccessful authentication attempts to make key search more difficult for attackers. Care should be taken so that this doesn't become a self-denial of service vector.

9.3.1 Weak Transport

If the transport layer does not provide confidentiality, authentication methods that rely on secret data SHOULD be disabled. If it does not provide strong integrity protection, requests to change authentication data (e.g. a password change) SHOULD be disabled to prevent an attacker from modifying the ciphertext without being noticed, or rendering the new authentication data unusable (denial of service).

The assumption as stated above that the Authentication Protocol only run over a secure transport that has previously authenticated the server is very important to note. People deploying SSH are reminded of the consequences of man-in-the-middle attacks if the client does not have a very strong a priori association of the server with the host key of that server. Specifically for the case of the Authentication Protocol the client may form a session to a man-in-the-middle attack device and divulge user credentials such as their username and password. Even in the cases of authentication where no user credentials are divulged, an attacker may still gain information they shouldn't have by capturing key-strokes in much the same way that a honeypot works.

9.3.2 Debug messages

Special care should be taken when designing debug messages. These messages may reveal surprising amounts of information about the host if not properly designed. Debug messages can be disabled (during user authentication phase) if high security is required. Administrators of host machines should make all attempts to compartmentalize all event notification messages and protect them from unwarranted observation. Developers should be aware of the sensitive nature of some of the normal event messages and debug messages and may want to provide guidance to administrators on ways to keep this information away from unauthorized people. Developers should consider minimizing the amount of sensitive information obtainable by users during the authentication phase in accordance with the local policies. For this reason, it is RECOMMENDED that debug messages be initially disabled at the time of deployment and require an active decision by an administrator to allow them to be enabled. It is also RECOMMENDED that a message expressing this concern be presented to the administrator of a system when the action is taken to enable debugging messages.

9.3.3 Local security policy

Implementer MUST ensure that the credentials provided validate the professed user and also MUST ensure that the local policy of the

server permits the user the access requested. In particular, because of the flexible nature of the SSH connection protocol, it may not be possible to determine the local security policy, if any, that should apply at the time of authentication because the kind of service being requested is not clear at that instant. For example, local policy might allow a user to access files on the server, but not start an interactive shell. However, during the authentication protocol, it is not known whether the user will be accessing files or attempting to use an interactive shell, or even both. In any event, where local security policy for the server host exists, it MUST be applied and enforced correctly.

Implementors are encouraged to provide a default local policy and make its parameters known to administrators and users. At the discretion of the implementors, this default policy may be along the lines of 'anything goes' where there are no restrictions placed upon users, or it may be along the lines of 'excessively restrictive' in which case the administrators will have to actively make changes to this policy to meet their needs. Alternatively, it may be some attempt at providing something practical and immediately useful to the administrators of the system so they don't have to put in much effort to get SSH working. Whatever choice is made MUST be applied and enforced as required above.

9.3.4 Public key authentication

The use of public-key authentication assumes that the client host has not been compromised. It also assumes that the private-key of the server host has not been compromised.

This risk can be mitigated by the use of passphrases on private keys; however, this is not an enforceable policy. The use of smartcards, or other technology to make passphrases an enforceable policy is suggested.

The server could require both password and public-key authentication, however, this requires the client to expose its password to the server (see section on password authentication below.)

9.3.5 Password authentication

The password mechanism as specified in the authentication protocol assumes that the server has not been compromised. If the server has been compromised, using password authentication will reveal a valid username / password combination to the attacker, which may lead to further compromises.

This vulnerability can be mitigated by using an alternative form of

authentication. For example, public-key authentication makes no assumptions about security on the server.

9.3.6 Host based authentication

Host based authentication assumes that the client has not been compromised. There are no mitigating strategies, other than to use host based authentication in combination with another authentication method.

9.4 Connection protocol

9.4.1 End point security

End point security is assumed by the connection protocol. If the server has been compromised, any terminal sessions, port forwarding, or systems accessed on the host are compromised. There are no mitigating factors for this.

If the client end point has been compromised, and the server fails to stop the attacker at the authentication protocol, all services exposed (either as subsystems or through forwarding) will be vulnerable to attack. Implementors SHOULD provide mechanisms for administrators to control which services are exposed to limit the vulnerability of other services.

These controls might include controlling which machines and ports can be target in 'port-forwarding' operations, which users are allowed to use interactive shell facilities, or which users are allowed to use exposed subsystems.

9.4.2 Proxy forwarding

The SSH connection protocol allows for proxy forwarding of other protocols such as SNMP, POP3, and HTTP. This may be a concern for network administrators who wish to control the access of certain applications by users located outside of their physical location. Essentially, the forwarding of these protocols may violate site specific security policies as they may be undetectably tunneled through a firewall. Implementors SHOULD provide an administrative mechanism to control the proxy forwarding functionality so that site specific security policies may be upheld.

In addition, a reverse proxy forwarding functionality is available, which again can be used to bypass firewall controls.

As indicated above, end-point security is assumed during proxy forwarding operations. Failure of end-point security will compromise

all data passed over proxy forwarding.

9.4.3 X11 forwarding

Another form of proxy forwarding provided by the ssh connection protocol is the forwarding of the X11 protocol. If end-point security has been compromised, X11 forwarding may allow attacks against the X11 server. Users and administrators should, as a matter of course, use appropriate X11 security mechanisms to prevent unauthorized use of the X11 server. Implementors, administrators and users who wish to further explore the security mechanisms of X11 are invited to read [SCHEIFLER] and analyze previously reported problems with the interactions between SSH forwarding and X11 in CERT vulnerabilities VU#363181 and VU#118892 [CERT].

X11 display forwarding with SSH, by itself, is not sufficient to correct well known problems with X11 security [VENEMA]. However, X11 display forwarding in SSHv2 (or other, secure protocols), combined with actual and pseudo-displays which accept connections only over local IPC mechanisms authorized by permissions or ACLs, does correct many X11 security problems as long as the "none" MAC is not used. It is RECOMMENDED that X11 display implementations default to allowing display opens only over local IPC. It is RECOMMENDED that SSHv2 server implementations that support X11 forwarding default to allowing display opens only over local IPC. On single-user systems it might be reasonable to default to allowing local display opens over TCP/IP.

Implementors of the X11 forwarding protocol SHOULD implement the magic cookie access checking spoofing mechanism as described in [ssh-connect] as an additional mechanism to prevent unauthorized use of the proxy.

Normative References

[SSH-ARCH]
Ylonen, T., "SSH Protocol Architecture", I-D draft-ietf-architecture-15.txt, Oct 2003.

[SSH-TRANS]
Ylonen, T., "SSH Transport Layer Protocol", I-D draft-ietf-transport-17.txt, Oct 2003.

[SSH-USERAUTH]
Ylonen, T., "SSH Authentication Protocol", I-D draft-ietf-userauth-18.txt, Oct 2003.

[SSH-CONNECT]

Ylonen, T., "SSH Connection Protocol", I-D draft-ietf-connect-18.txt, Oct 2003.

[SSH-NUMBERS]
Lehtinen, S. and D. Moffat, "SSH Protocol Assigned Numbers", I-D draft-ietf-secsh-assignednumbers-05.txt, Oct 2003.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Informative References

[FIPS-186]
Federal Information Processing Standards Publication, "FIPS PUB 186, Digital Signature Standard", May 1994.

[FIPS-197]
National Institue of Standards and Technology, "FIPS 197, Specification for the Advanced Encryption Standard", November 2001.

[ANSI T1.523-2001]
American National Standards Insitute, Inc., "Telecom Glossary 2000", February 2001.

[SCHEIFLER]
Scheifler, R., "X Window System : The Complete Reference to Xlib, X Protocol, Icccm, Xlfd, 3rd edition.", Digital Press ISBN 1555580882, Feburary 1992.

[RFC0854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.

[RFC0894] Hornig, C., "Standard for the transmission of IP datagrams over Ethernet networks", STD 41, RFC 894, April 1984.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1134] Perkins, D., "Point-to-Point Protocol: A proposal for multi-protocol transmission of datagrams over Point-to-Point links", RFC 1134, November 1989.

[RFC1282] Kantor, B., "BSD Rlogin", RFC 1282, December 1991.

[RFC1510] Kohl, J. and B. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", RFC 1700, October 1994.

[RFC1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[RFC3066] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[RFC2025] Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.

[RFC2085] Oehler, M. and R. Glenn, "HMAC-MD5 IP Authentication with Replay Prevention", RFC 2085, February 1997.

[RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2246] Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[RFC2410] Glenn, R. and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, November 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[SCHNEIER] Schneier, B., "Applied Cryptography Second Edition: protocols algorithms and source in code in C", 1996.

[KAUFMAN,PERLMAN,SPECINER] Kaufman, C., Perlman, R. and M. Speciner, "Network Security: PRIVATE Communication in a PUBLIC World", 1995.

[CERT] CERT Coordination Center, The., "http://www.cert.org/nav/

index_red.html".

[VENEMA] Venema, W., "Murphy's Law and Computer Security", Proceedings of 6th USENIX Security Symposium, San Jose CA http://www.usenix.org/publications/library/proceedings/sec96/venema.html, July 1996.

[ROGAWAY] Rogaway, P., "Problems with Proposed IP Cryptography", Unpublished paper http://www.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt, 1996.

[DAI] Dai, W., "An attack against SSH2 protocol", Email to the SECSH Working Group ietf-ssh@netbsd.org ftp://ftp.ietf.org/ietf-mail-archive/secsh/2002-02.mail, Feb 2002.

[BELLARE,KOHNO,NAMPREMPRE] Bellaire, M., Kohno, T. and C. Namprempre, "Authenticated Encryption in SSH: Fixing the SSH Binary Packet Protocol", , Sept 2002.

Authors' Addresses

Tatu Ylonen
SSH Communications Security Corp
Fredrikinkatu 42
HELSINKI FIN-00100
Finland

EMail: ylo@ssh.com

Darren J. Moffat (editor)
Sun Microsystems, Inc
17 Network Circle
Menlo Park CA 94025
USA

EMail: Darren.Moffat@Sun.COM

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

บรรณานุกรม

หนังสืออ้างอิงภาษาอังกฤษ

[1] Daniel J. Barrett and Richard E. Silverman. "SHH, the Secure Shell: The Definitive Guide", O'Reilly & Associates, Inc. 2001.

[2] "TCP/IP Tutorial and Technical Overview", Redbooks, IBM Corporation. 2001

หนังสืออ้างอิงภาษาไทย

[1] เชาวนิน ไสยสมบัติ และสุชาติ คุ้มมะณี. "โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวา". ปริญญานิพนธ์ปีการศึกษา 2542. ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สจล. 2542

เวปเพจอ้างอิง

[1] ANSI/VT100 Terminal Control
http://www.fh-jena.de/~gmueller/Kurs_halle/esc_vt100.html

[2] The Telnet Protocol
http://www.scit.wlv.ac.uk/~jphb/comms/telnet.html

[3] Secure Shell (secsh) Charter - SSH2 Internet-Drafts Standard
http://www.ietf.org/html.charters/secsh-charter.html

[4] CryptLib
http://www.cs.auckland.ac.nz/~pgut001/cryptlib/

[5] NetCat
http://www.atstake.com/research/tools/network_utilities/

[6] SHH, the Secure Shell: The Definitive Guide
http://www.snailbook.com

[7] Microsoft Developer Network (MSDN)
http://msdn.microsoft.com