

KEYSTROKE ATTACK ON SSH

By

Michael Lustig and Yonit Shabtai

Supervisor

Yoram Yihyie

Final Project Report

AT

TECHNION IIT

ELECTRICAL ENGINEERING

COMPUTER NETWORKS LABORATORY

TECHNION CITY, HAIFA 32000 ISRAEL

OCTOBER 2001

© Copyright by Michael Lustig and Yonit Shabtai, 2001

Table of Contents

Table of Contents	ii
List of Figures	iv
Introduction	1
1 SSH Overview	2
1.1 SSH1	2
1.1.1 Encryption:	3
1.1.2 Integrity:	3
1.1.3 Key-exchange:	4
1.1.4 Authentication:	4
1.2 SSH2	5
1.2.1 Transport Layer:	5
1.2.2 User Authentication Layer:	6
1.2.3 Connection Layer:	7
2 Attack on SSH	8
2.1 SSH weaknesses	8
2.2 Keystroke Characteristics	9
2.3 Practical Attacks on Passwords	9
2.4 Keystroke timings as a Hidden Markov Model	10
2.5 Solving the HMM - the n-Viterbi Algorithm	11
3 The Attack System: description and user-guide	13
3.1 The Attack System Structure	13
3.2 The Keystrokes Statistics Generator	14
3.3 The Sniffer Module	16
3.4 The Parser Module	19

3.5	The n-Viterbi Module	19
3.6	The Test Module	20
4	The Attack System: Results and Conclusions	21
4.1	Keystroke timing test	21
4.2	Attack Results and conclusions	25
	Bibliography	27

List of Figures

1.1	SSH1 packet structure	3
1.2	SSH2 layered architecture	5
1.3	SSH2 packet structure	6
2.1	SU attack	10
2.2	The n-Viterbi algorithm	12
3.1	The attack system schematics	14
3.2	The Ethereal window setup	17
3.3	The ethereal edit menu	18
3.4	The ethereal preferences window	18
3.5	The sniffer parser state-machine	19
4.1	The histogram of inter keystroke latencies.	22
4.2	The functions of the keystroke latencies.	22
4.3	The entropy of key pairs with knowledge of the inter keystroke latency.	24
4.4	The amount of information learned on key pairs with knowledge of the inter keystroke latency.	25

Introduction

SSH is a protocol for secure network transmission. It is intended for replacing the unsecured protocols like Telnet,rlogin,rsh,ftp etc. The protocol uses cryptographic authentication of both user and server, automatic session encryption and integrity protection for the transmitted data. There are two different protocols: SSH1 and SSH2, which differ in both their architecture and the cryptographic algorithms they use.

Although using state of the art encryption algorithms, SSH has two weaknesses: First, packets are padded to eight-byte boundary, which helps inferring on short passwords. Second, in interactive mode, a packet is sent on every keystroke, which leaks the inter keystroke timing. We have conducted experiments which show that keystroke timing leaks a considerable amount of information about what was typed. We have developed a system which can infer user's passwords from keystroke timing.

The outline of this report is as follows: in chapter 1 we will give an overview of the SSH protocol. In chapter 2 we will describe the theoretical background and the attack on SSH. in chapter 3 we will give a complete description of the system developed and in chapter 4 we will describe the experimental results and the conclusions.

Chapter 1

SSH Overview

As mentioned briefly in the introduction, SSH ([2],[3],[6],[7], [9]) is a protocol for secure network transmission. It was first intended to replace the unsecured protocols such as Telnet, rlogin, rsh etc. SSH uses cryptographic authentication, session encryption and integrity protection for the transferred data. According to [3] *"SSH is being used by thousands of sites in at least 50 countries. Its users include top universities, research labs and major corporations."*

There are two different protocols: SSH1 and SSH2.

1.1 SSH1

SSH1 is a packet based binary protocol, that works on top of any transport layer, normally TCP/IP. The packet mechanism, the authentication, key exchange, encryption and integrity mechanism implement a secured transport layer, which is used to implement the remote-login application.

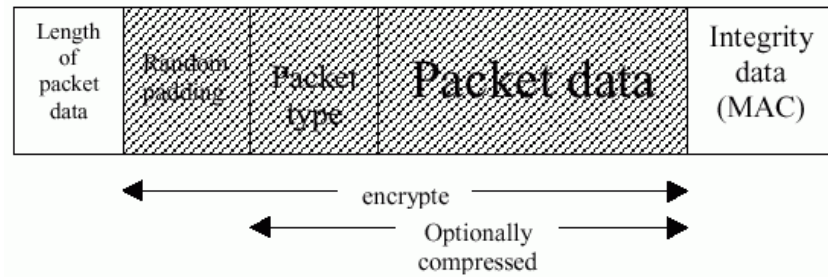


Figure 1.1: SSH1 packet structure

1.1.1 Encryption:

A SSH1 packet consists of:

1. length of packet data
2. random padding
3. packet type
4. packet data
5. integrity protection data(MAC)

Packet encryption is done using block cyphers such as IDEA-CFB, 3DES-CBC etc. Items 3,4 can be compressed using gzip before encryption. the packet length is rounded to multiple of eight bytes boundaries in such a way that the random padding length is 1-8 bytes long.

1.1.2 Integrity:

The integrity of the data is provided by CRC32, or HMAC-SHA1 on the plaintext. If tampering is detected, the error is logged, the user is notified and the connection

is closed.

1.1.3 Key-exchange:

The key exchange mechanism in SSH1 is based on RSA.

- The server sends its public key and another RSA key that changes every hour.
- The client produces a 256 bit random number session key, and chooses an encryption algorithm.
- The client encrypts the session key using the RSA keys, and sends it to the host.
- The host decrypts the session key using its private keys, and then encrypts a confirmation using the session key. That way the client knows that the host has the session key.

1.1.4 Authentication:

HOST: The host's public key is its authentication. The client must have prior knowledge of the host's key. The client compares the host key with its own database. There is a possibility to use CA. USERS: user authentication can be done in two different ways: The first is password authentication, and the second, by public key algorithm such as RSA. The host challenges the client with a random number encrypted using the client's public key.

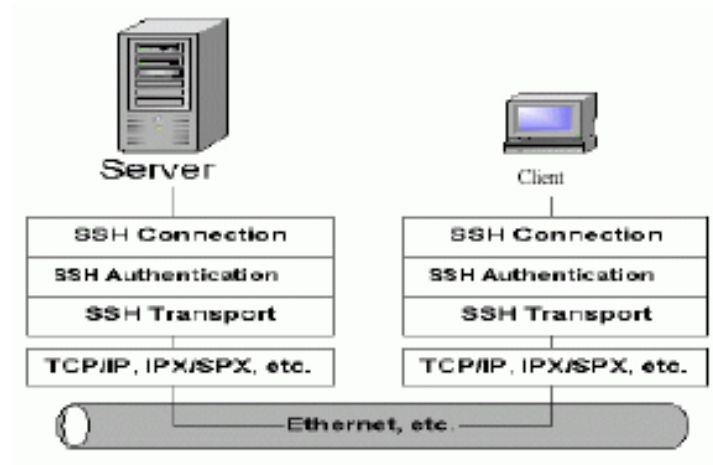


Figure 1.2: SSH2 layered architecture

1.2 SSH2

SSH2 protocol is designed in a new layered architecture to provide modularity, and better security. There are three layers: *Transport Layer* provides encryption, integrity and server authentication. *User Authentication Layer* provides the user authentication. *Connection Layer* runs applications such as interactive login.

1.2.1 Transport Layer:

The transport layer provides the host authentication. All algorithms are negotiated at the start of the session. After this, the key is exchanged using Diffie-Helman method. The authentication of the host is done using RSA/DSS signatures. Encryption is achieved by the use of CBC cyphers such as: 3DES, Blowfish, Twofish, Idea etc. The integrity of the data is achieved by MAC, the MAC is calculated after compression and before encryption. The MAC algorithms used are HMAC-SHA1 and HMAC-MD5.

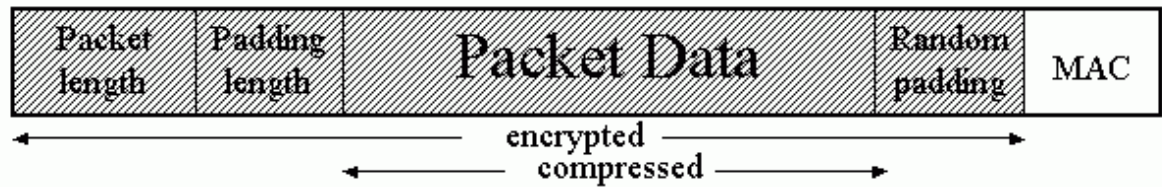


Figure 1.3: SSH2 packet structure

The SSH2 packet consists of:

1. Packet length
2. Padding Length
3. Data
4. Random Padding
5. MAC

In SSH2 the packet length field is also encrypted, as shown in Fig. 1.3, which increases security.

1.2.2 User Authentication Layer:

This layer, runs over the transport layer, the integrity and confidentiality are assumed. The user authentication can be preformed in two ways: The first is using public key authentication, and the second is by password authentication.

1.2.3 Connection Layer:

This layer runs the applications concerning SSH for example: Interactive login, remote exec, X11 & TCP forwarding and multiplexing of multiple channels into one session.

Chapter 2

Attack on SSH

In this chapter we will discuss the SSH weaknesses, and how they can be used to implement an attack on the SSH protocol. We will also describe the theoretical background, and the algorithms used for the attack.

2.1 SSH weaknesses

Despite state of the art encryption algorithms, SSH has seemingly minor two weaknesses:

First, passwords are padded to an eight byte boundary. This weakness leaks the approximated length of the password, for example, a seven byte password will be sent in an eight byte packet, whereas a longer password will be sent in a 16 bytes packet or longer. Knowing this, short passwords can be inferred and attacked.

Second, in interactive mode, in every keystroke a packet is sent. This weakness leaks the inter keystroke timing of the user, since the arrival time of packets is almost the same as the inter keystroke timing. An eavesdropper can measure the arrival time of packets of an SSH session. It was already shown in many works (see ref. in [1])

that keystroke timing leaks information about the identity of the user. Moreover, it leaks information about what was typed.

Next we will describe, how by using these two weaknesses, a practical attack on SSH can be implemented, and cause serious security faults. We will describe how users passwords can be inferred from the inter keystroke timings.

2.2 Keystroke Characteristics

By examining the keystrokes characteristics, we can see that different character pairs have different latencies. We used the model suggested by [1], which is a gaussian model of the inter keystroke timing. Each character pair has a distribution such that

$$P(q|y) \sim N(\mu_q, \sigma_q^2) \quad (2.2.1)$$

Where q is the character pair, y is the timing measured, and μ_q and σ_q are the corresponding normal distribution parameters.

This model can be built by gathering statistics, and calculating the mean and variance of the keystroke timing.

2.3 Practical Attacks on Passwords

A keystroke attack will be effective only when a short sequence of keystroke, which is important, can be inferred from the timing. Passwords qualify to this description. [1] suggests two practical attacks to capture keystroke timings of a user's password.

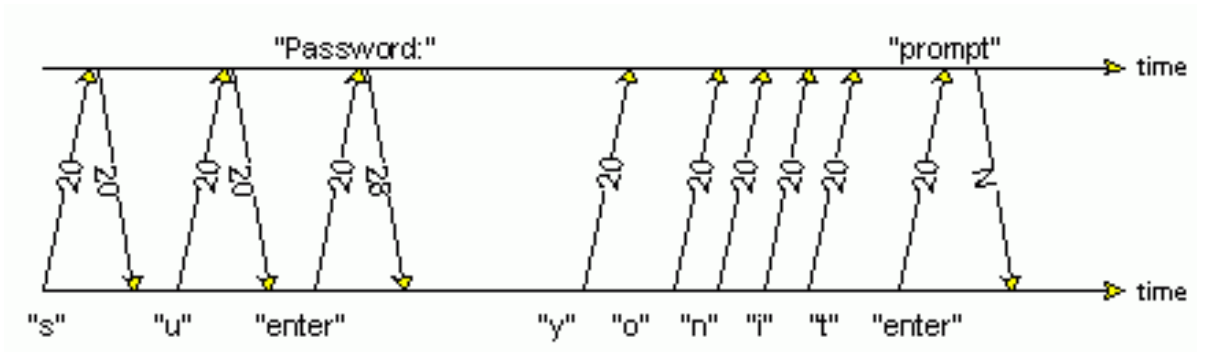


Figure 2.1: SU attack

SU attack Fig. 2.1 shows an SU command in a SSH session. Specifically this is the "signature" of SSH1 using 3DES encryption, which is a very common setup of SSH. The user types S, and receives the echo. Then the user types U and receives the echo, the user then strikes the "enter" key, and a packet containing the prompt "password:" with length 28 bytes is sent back. Now, when the user types the password, no echo is sent back, and an eavesdropper can measure the arrival time of the password keystrokes packets, and get the inter keystroke timing of the super user's password. In this work we implemented this attack. For other attacks please refer to [1]

2.4 Keystroke timings as a Hidden Markov Model

In section 2.3 we discuss an attack in which we can measure the keystroke timing of a super-user password. In order to infer from the timings what was typed we model keystroke timing latencies as a hidden Markov model as suggested by [1].

Hidden Markov Model: A Markov process is a finite state stochastic process, in which the probability of transition from the current state to the next state depends only on the current state. A HMM is a Markov process, in which the current state is

not known, but the outputs of the process can be observed. In a HMM the probability of the outputs depends only on the current state and information about prior path of the process can be inferred from the outputs.

The inter keystroke timing is modelled as a HMM in the following way: Each character pair q is a hidden state in the process, and the keystroke timing y is the output observation of the process, such that the distribution in (2.2.1) is met. Our motivation is that efficient algorithm to working with HMM's exist. For more information on HMM see [4]

2.5 Solving the HMM - the n-Viterbi Algorithm

The Viterbi algorithm [5] is widely used in solving HMM's. The algorithm finds the most likely sequence in an efficient dynamic programming method. Let

- (y_1, \dots, y_t) the observations of the process
- (q_1, \dots, q_t) the most likely character pairs
- $S(q_t)$ the most likely sequence, ending with q_t with posteriori probability $V(q_t)$.

The algorithm goes as follows

- Init: $V(q_1) = P(q_1|y_1)$
- Iterate: $V(q_t) = \max_{q_{t-1}} P(y_t|q_t)P(q_t|q_{t-1})V(q_{t-1})$
- $S(q_t) = \arg \max_{q_{t-1}} P(y_t|q_t)P(q_t|q_{t-1})V(q_{t-1})$

But, because in our case the probability distributions are dense, the most likely sequence will probably not give the correct sequence. As suggested by [1], rather than

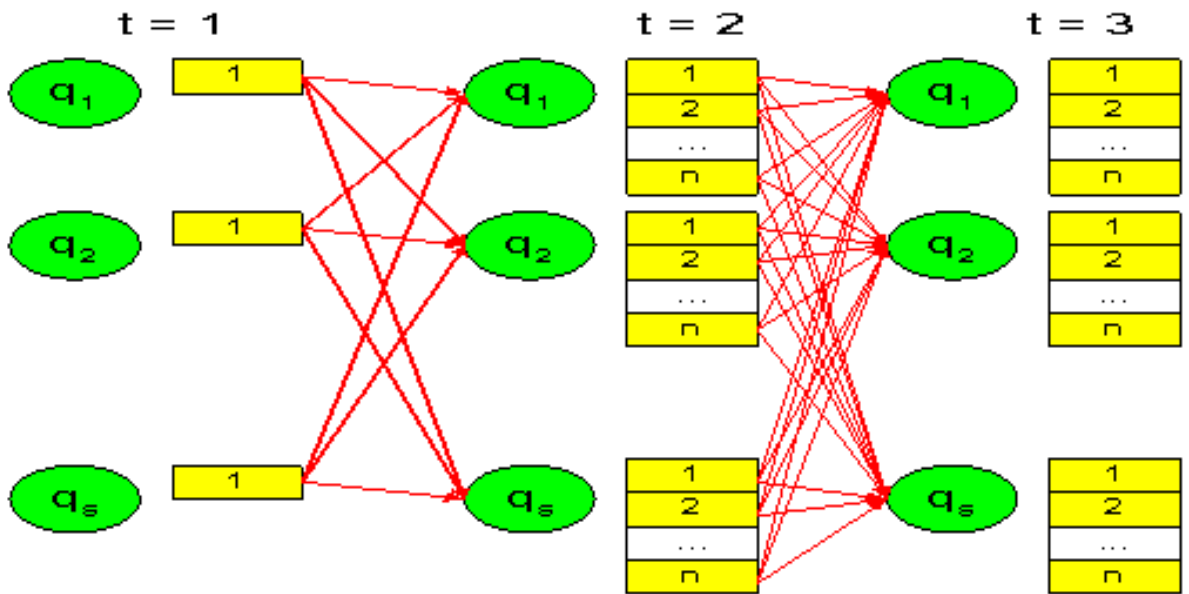


Figure 2.2: The n-Viterbi algorithm

using the Viterbi algorithm, we use a modification to the algorithm that returns the n most likely sequences. In every iteration, instead of saving the most likely sequence, we save the n most likely sequences as in Fig. 2.2.

Chapter 3

The Attack System: description and user-guide

In this chapter we will give an extensive description of the project. We will describe the architecture of the attack system, and of each module by itself. We will also give instructions on how to use our system.

3.1 The Attack System Structure

Figure 3.1 describes the attacking system. Our system is consisted of four main modules

The Sniffer: This module eavesdrops the network and records the network's traffic to a file. It then filters the recording and outputs the SSH traffic alone.

The Statistic Generator: This module gathers users inter keystroke timing statistics.

The Parser: This module processes the sniffer's output file, searches for a SU command, and extracts the password packets arrival-time intervals.

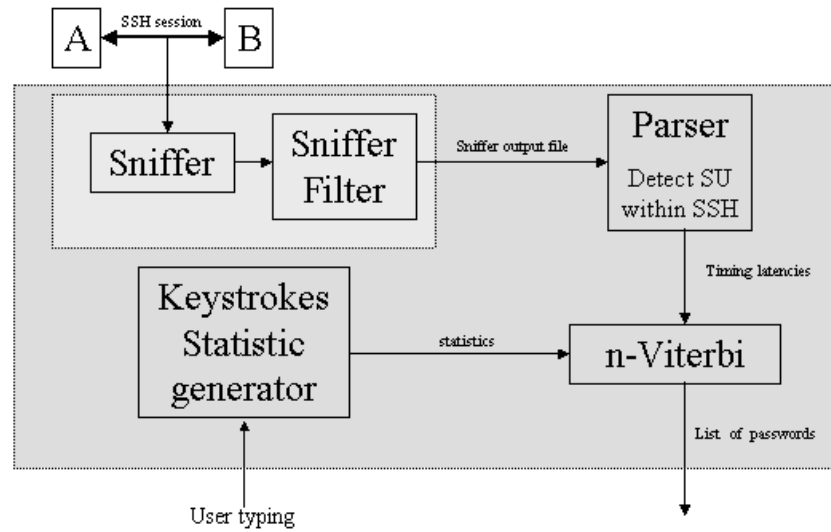


Figure 3.1: The attack system schematics

The n-Viterbi: This module processes the timings and produces a list of the n most likely passwords according to the statistics.

The modules are independent, and not automated. This means that they should be activated manually in the correct order for the full attack.

3.2 The Keystrokes Statistics Generator

This is the tool that gathers the keystrokes statistics and creates the statistics file. When started, the application searches for previous statistics files. If found, the user is prompt to choose whether to create new statistics files and override the existing, or to continue a previous session. Finally, the user is requested to start typing. The application records the inter-keystroke timing and the characters typed. It stores the data in a file. When finished typing, the application uses the timing data to create the statistics file.

The application uses two auxiliary files in order to save intermediate data and to provide the ability to continue a previous training session. The two files are the *symbol.dat* file, and the *timing.log* file.

In the application, only typed symbols during training will have an entry in the statistics file. The *symbol.dat* file is a binary file with the following structure:

- int numSymbols [size: 4 bytes]
- char symbolTable[numSymbols] [size: numSymbols bytes]

Whenever we strike a new character during the training, the character is added to the symbol table, and written to *symbol.dat* file. For example, if we strike the following sequence of characters: bbabdaf , the resulting *symbol.dat* file will be:

- numSymbols = 4
- symbolTable = "badf"

The other auxiliary file is the *timing.log*. In this file we record the inter keystroke timing measured for a character pair. *timing.log* is a text-file in which each line represents one character pair of keystrokes in the following way:

first key second key timing

In the same example, the sequence of characters: bbabdaf ,with the corresponding timing: 230.32, 240.41, 320.33, 420.23, 380.11, 110.97 will produce the following *timing.log*:

```

0 0 230.32
0 1 240.41
1 0 320.33
0 2 420.33
2 1 380.11
1 3 110.97

```

In *symbol.dat* file, we have the characters that correspond to the symbols numbers. For example, $0 = b$, $1 = a$, $2 = d$ and $3 = f$. It is important to note that large latencies of keystroke timing are ignored and not included in the *timing.log* file, because they represent breaks in typing rather than keystroke latencies.

The auxiliary files are used by the application to calculate the statistics needed for the n-Viterbi module. The application uses the timing file, to find the mean value μ and variance σ of the timing latencies y of each character pair q , that was typed during training. This data is written into the *statistics.dat* file. The file is a binary file that contains the probability transition matrix where index (i, j) represents the keystroke probability distribution $P(y|q) \sim N(\mu, \sigma)$ of the character pair i, j . The matrix is sized $numSymbols \times numSymbols$ and is a $2D$ array of the following structure:

- double μ
- double σ

This $2D$ array is the input to the n-Viterbi algorithm.

3.3 The Sniffer Module

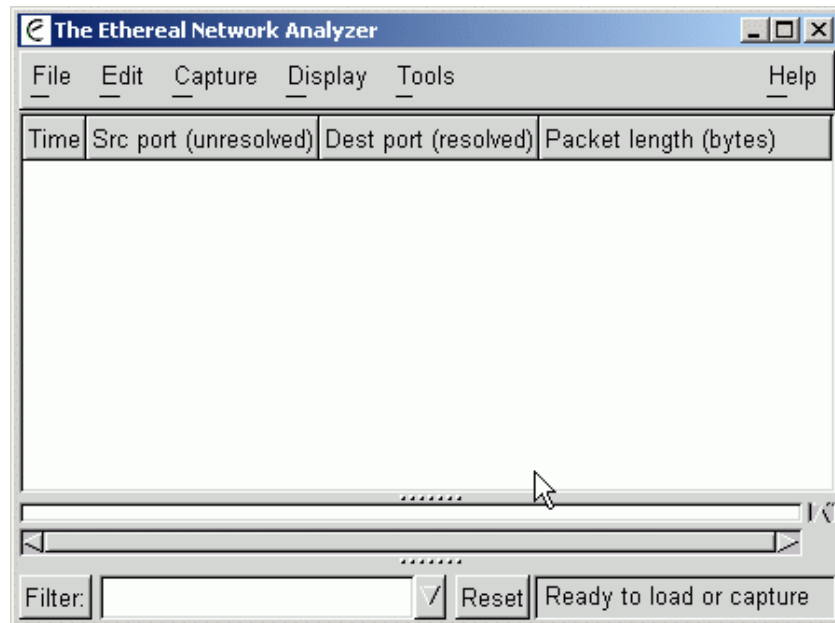


Figure 3.2: The Ethereal window setup

We used the Ethereal sniffer, but any sniffer that can output the file format needed for the parser is adequate. Ethereal should be installed, and configured as shown in Fig. 3.2. The fields shown should be, the arrival time of packets, the source port, the destination port, and the packet length. To setup this configuration, go to the edit menu and press preferences (Fig. 3.3). The Ethereal preferences window will appear. Switch to the "Columns" tab (Fig. 3.4) and set the columns as described in the figure.

To start the attack the sniffer should be activated to record the network traffic. In order to create the appropriate file for our parser program, the session should be saved as an ethereal file, and then the following command line should be applied:

```
tethereal.exe -r "ethreal file-name" -f "tcp.port==22 && frame.pkt_len>70"
```

The output should be redirected to a file using the redirect operator (>). After this

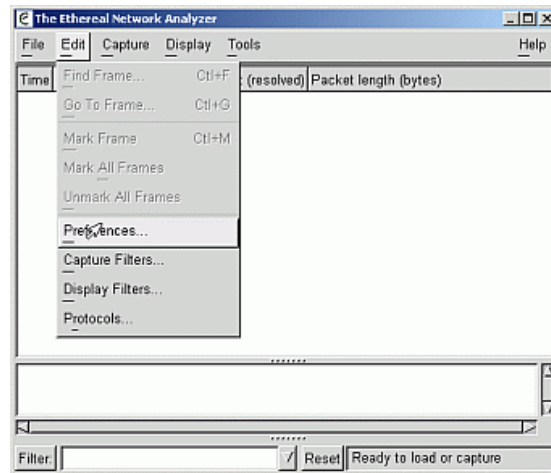


Figure 3.3: The ethereal edit menu

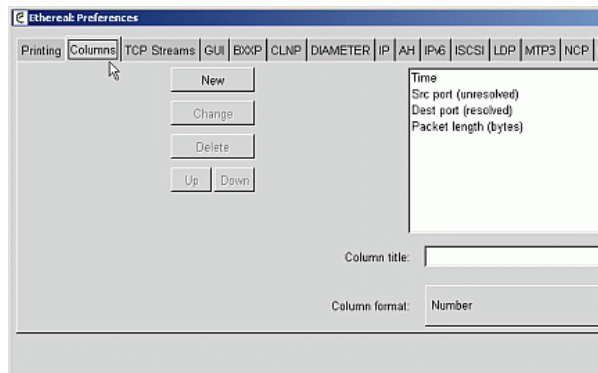


Figure 3.4: The ethereal preferences window

command a text-file is created, in the following format:

```
time arrival(double) source port(int) destination port(int) packet length(int)
```

This file is used as input to the parser module.

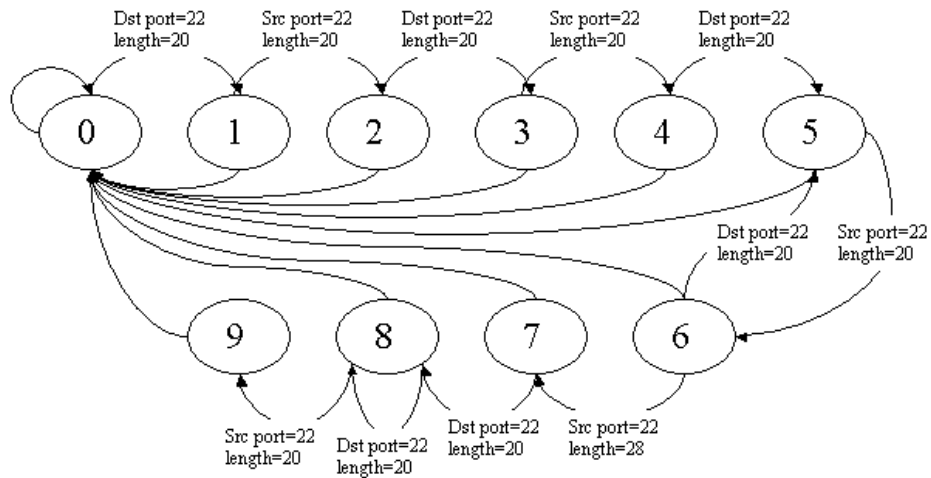


Figure 3.5: The sniffer parser state-machine

3.4 The Parser Module

This is the tool that processes the sniffer output file to find a SU command within a SSH session, and then extracts the keystroke timing of the super-user's password. The application is a simple finite state machine that searches for a fingerprint as in Fig 3.5. The command line is:

```
process_sniffer_output.exe snifferFile keystrokeTimingFile
```

where *snifferFile* is the output file of the sniffer, and *keystrokeTimingFile* is a text-file containing the timing extracted from the sniffer file, in float format.

3.5 The n-Viterbi Module

This module is the implementation of the n-Viterbi algorithm. It receives the statistics and the keystroke timing files as input, and produces a list of passwords starting with

the most likely one. The command line is:

processTiming.exe statisticsFile keystrokeTimingFile

The application, if encountered with a large latency, divides the timings into phrases and tries to guess each phrase separately.

3.6 The Test Module

This module is for test purposes only. It receives the statistics file as input, and the user is asked to type. The application measures the inter keystroke timing, uses the n-Viterbi to find the most likely sequences, and checks the location of the right guess. This tool is intended to check the statistics gathered. If the statistics is good, the actual sequence will be located among the first percentage of the n-Viterbi's guessed list. The command line is:

guess_keystrokes.exe statisticsFile symbolFile

Chapter 4

The Attack System: Results and Conclusions

4.1 Keystroke timing test

In order to learn about the statistical properties of the inter keystroke latencies, we conducted an experiment. We randomly chose four letter keys, two number keys, and two upper-case keys (i,a,k,m,2,3,O,J). Using these keys we formed 64 pairs. Using the statistics generator tool, a user was asked to type each pair 30 times. The mean value and standard deviation was calculated for each pair. Figure 4.1 shows the histogram of the keystroke timings. We can see that there exist three isolated groups of timings. The first group, can be divided by two, the faster timings refer to close letter and number key pairs that were typed using one hand. The slower part of the first group refers to letter and number key pairs typed with both hands, which enables parallelism and still fast timing. The second group refers to key pairs that contain both letter

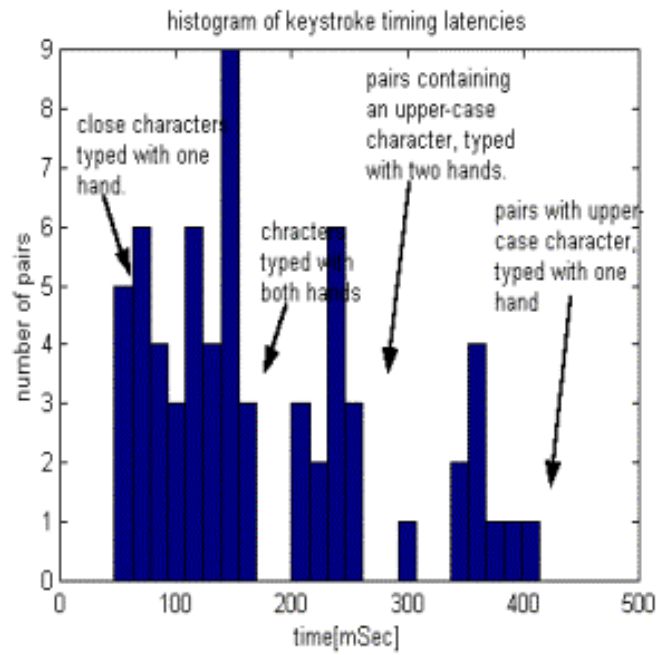


Figure 4.1: The histogram of inter keystroke latencies.

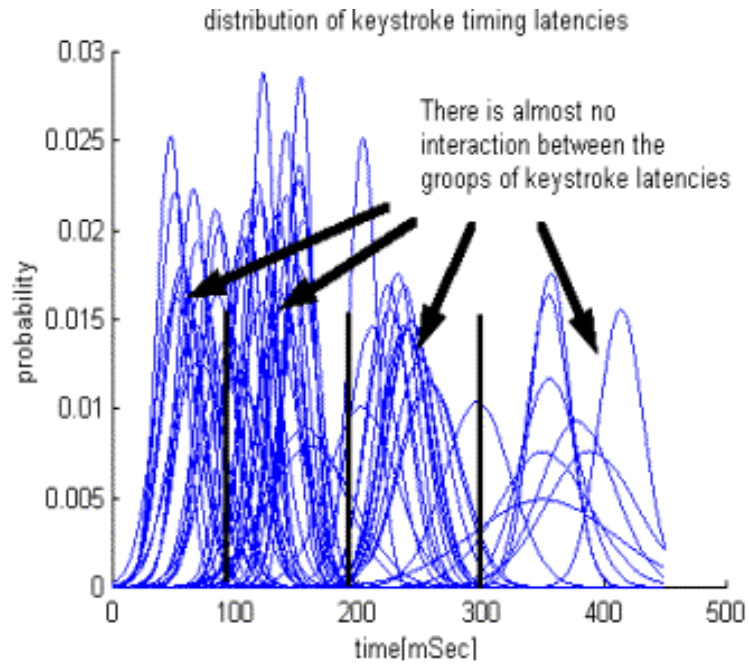


Figure 4.2: The functions of the keystroke latencies.

and upper-case letters that were typed with both hands. The slowest timing is to a combination of upper-lower case keys that were typed using one hand. From the graph we learn that by knowing the inter keystroke timing, we can distinguish easily between these groups. If we model the keystroke timing as a gaussian distribution, we can see from Figure 4.2, that there is almost no overlapping of the distributions of the four different kind of key pairs, and they can be distinguished easily. For example, if we see a keystroke timing of 380[ms], it is probably a key pair of both upper and lower case letters typed with the same hand.

We would like to quantify the amount of information that can be learned from the inter keystroke timing. An attacker without prior knowledge can guess any key pairs q out of the set Q with the same probability, thus the entropy of the key pairs in our experiment (a set of 64 pairs) is

$$H_0[q] = - \sum_{q \in Q} Pr(q) \log_2[Pr(q)] = \log_2[|Q|] = 6_{[bits]}, \quad (4.1.1)$$

The entropy of key pairs for an attacker that knows the inter keystroke timing y_0 is

$$H_1[q|y = y_0] = - \sum_{q \in Q} Pr(q|y = y_0) \log_2[Pr(q|y = y_0)], \quad (4.1.2)$$

But, we know the probability $Pr(y_0|q)$, so by using Bayes theorem we can get

$$Pr(q|y_0) = \frac{Pr(y_0|q)Pr(q)}{Pr(y_0)} = \frac{Pr(y_0|q)Pr(q)}{\sum_{\tilde{q} \in Q} Pr(y_0|\tilde{q})Pr(\tilde{q})}$$

And this can be computed from the experiment results.

In Fig. 4.3 describes the entropy of key pairs with knowledge of the inter keystroke timing. Note, that in timings that have dense probabilities curves, the entropy is high because it is harder to guess the key pairs from the timings. Fig 4.4 shows the amount

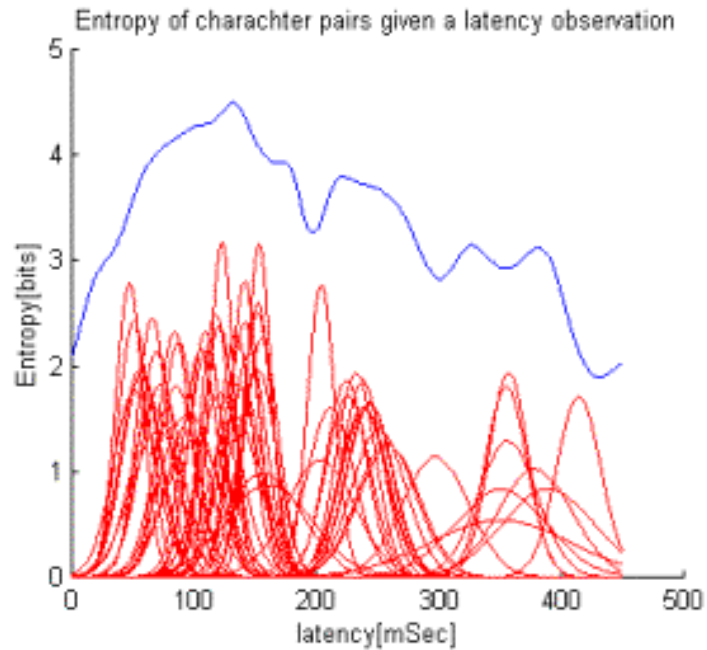


Figure 4.3: The entropy of key pairs with knowledge of the inter keystroke latency.

of information learned. And the total amount of information gain is

$$\begin{aligned}
 I &= H_0[q] - H_1[q|y] = H_o[q] - \int Pr(y_0)H_1[q|y_0] dy_0 = \\
 &= 6_{[bits]} - 2.6853_{[bits]} = 3.3147_{[bits]}.
 \end{aligned}$$

Although these results, in our opinion, are quite "optimistic", because we used a small set of key pairs. It is shown that inter keystroke can leak a considerable amount of information.

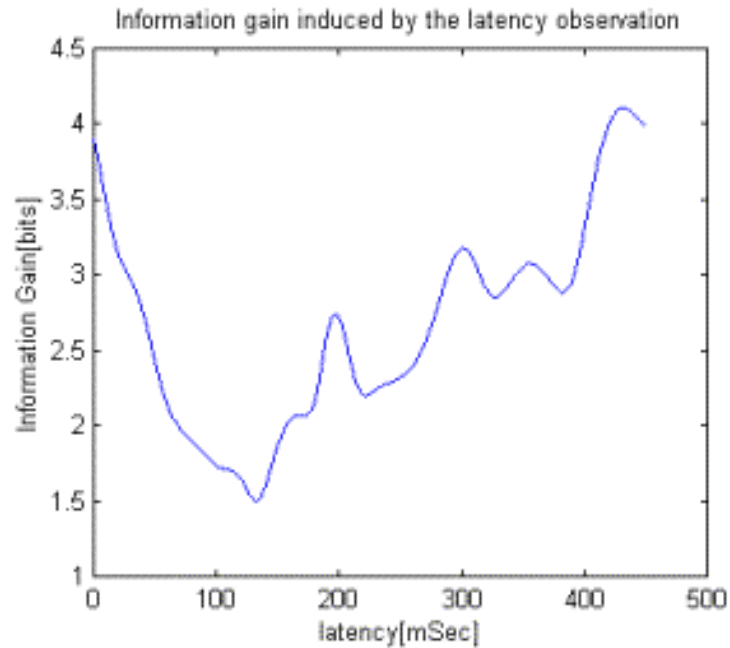


Figure 4.4: The amount of information learned on key pairs with knowledge of the inter keystroke latency.

4.2 Attack Results and conclusions

As in [1], we conducted an experiment to find out how good our system guess passwords from inter keystroke timings. From the above characters, we chose three passwords. The attack system was asked to guess these passwords in two conditions. The first condition was that the passwords were typed by the same user that the statistics were gathered from, and the second was that the passwords were typed by another person. In order to scale the quality of the result, we found the location of the real password in the guessed passwords list, and then calculated the percentage of the location. Note that with no prior information, the average percentage is 50%. The Table below describes the results

	Pass1	Pass2	Pass3
User1	12%	1%	3%
User2	16%	4%	2%

The results above show that using the inter keystroke timing information, reduces the full search by 4 to 500 times. It shows that the keystroke attack is practical and is a serious security breach in the SSH protocol. It is also shown that a training set of one user can be used effectively on another user with good results.

By studying keystrokes, we learned that the simple Markov model does not describe the inter keystroke timing accurately. For example, if someone types the three letters "abh" the timing of the pair "bh" will not be the same as in typing "bha". We think, that in expanding the model to a second degree Markov model, the results will be much better.

Bibliography

- [1] "Timing Analysis of Keystrokes and Timing Attacks on SSH" Dawn Xiaodong Song, David Wagner, and Xuqing Tian. 10th USENIX Security Symposium, 2001.
- [2] The project home-page, <http://comnet.technion.ac.il/cn19s01>
- [3] T. Ylmen "SSH-Secure Login Connection over the Internet, SSH Communications Security Ltd.
- [4] Y. Ephraim and, N. Merhav, "Hidden Markov Processes" , CCIT Report #322, August 2000.
- [5] G. D. Forney, Jr., "The Viterbi algorithm," Proc. IEEE, vol. 61, pp. 268-278, Mar. 1973.
- [6] <http://www.ssh.com>
- [7] <http://www.openssh.com>
- [8] <http://www.ethereal.com>
- [9] SSH protocol internet draft, <http://www.ssh.com/tech/archive/secsh.cfm>