Compromising Electromagnetic Emanations of Wired and Wireless Keyboards

Martin Vuagnoux
LASEC/EPFL
martin.vuagnoux@epfl.ch

Sylvain Pasini
LASEC/EPFL
sylvain.pasini@epfl.ch

Abstract

Computer keyboards are often used to transmit confidential data such as passwords. Since they contain electronic components, keyboards eventually emit electromagnetic waves. These emanations could reveal sensitive information such as keystrokes. The technique generally used to detect compromising emanations is based on a wide-band receiver, tuned on a specific frequency. However, this method may not be optimal since a significant amount of information is lost during the signal acquisition. Our approach is to acquire the raw signal directly from the antenna and to process the entire captured electromagnetic spectrum. Thanks to this method, we detected four different kinds of compromising electromagnetic emanations generated by wired and wireless keyboards. These emissions lead to a full or a partial recovery of the keystrokes. We implemented these side-channel attacks and our best practical attack fully recovered 95% of the keystrokes of a PS/2 keyboard at a distance up to 20 meters, even through walls. We tested 12 different keyboard models bought between 2001 and 2008 (PS/2, USB, wireless and laptop). They are all vulnerable to at least one of the four attacks. We conclude that most of modern computer keyboards generate compromising emanations (mainly because of the manufacturer cost pressures in the design). Hence, they are not safe to transmit confidential information.

1 Introduction

Today, most of the practical attacks on computers exploit software vulnerabilities. New security weaknesses are disclosed every day, but patches are commonly delivered within a few days. When a vulnerability is based on hardware, there is generally no software update to avoid the exposure: the device must be changed.

Computer keyboards are often used to transmit sensitive information such as passwords, e.g. to log into computers, to do e-banking money transfer, etc. A weakness in these hardware devices will jeopardize the security of any password-based authentication system.

Compromising electromagnetic emanation problems appeared already at the end of the 19^{th} century. Because of the extensive use of telephones, wire networks became extremely dense. People could sometimes hear other conversations on their phone line due to undesired coupling between parallel wires. This unattended phenomenon, called *crosstalk*, may be easily canceled by twisting the cables.

A description of some early exploitations of compromising emanations has been recently declassified by the National Security Agency [26]. During World War II, the American Army used teletypewriter communications encrypted with Bell 131-B2 mixing devices. In a Bell laboratory, a researcher noticed, quite by accident, that each time the machine stepped, a spike appeared on an oscilloscope in a distant part of the lab. To prove the vulnerability of the device, Bell engineers captured the compromising emanations emitted by a Bell 131-B2, placed in a building across the street and about 25 meters away. They were able to recover 75% of the plaintext.

During the Vietnam war, a sensor called *Black Crow* carried aboard C-130 gunships was able to detect the electromagnetic emanations produced by the ignition system of trucks on the Ho Chi Minh trail, from a distance up to 10 miles [25, 11].

1.1 Related Work

Academic research on compromising electromagnetic emanations started in the mid 1980's and there has been significant recent progresses [28, 1]. The threat related to compromising emanations has been constantly confirmed by practical attacks such as Cathode Ray Tubes (CRT) displays image recovery [34], Liquid Crystal Display (LCD) image recovery [20], secret key disclosure [16], video displays risks [18, 33] or radiations from

FPGAs [24].

Compromising electromagnetic emanations of serial-port cables have been already discussed by Smulders [30] in 1990. PS/2 keyboards still use bi-directional serial communication to transmit the pressed key code to the computer. Hence, some direct compromising electromagnetic emanations might appear. However, the characteristics of the serial line changed since the 90's. The voltage is not 15 volts anymore and the transition times of the signals are much longer (from picoseconds to microseconds).

Since keyboards are often the first input device of a computer system, they have been intensively studied. For instance, the exploitation of visual compromising information leaks such as optical reflections [5] which could be applied to keyboards, the analysis of surveillance video sequences [6] which can be used by an attacker to recover the keystrokes (even with a simple webcam) or the use of the blinking LEDs of the keyboard as a covert channel [21]. Acoustic compromising emanations from keyboards have been studied as well. Asonov and Agrawal [4] discovered that each keystroke produces a unique sound when it is pressed or released and they presented a method to recover typed keystrokes with a microphone. This attack was later improved, see [38, 7]. Even passive timing analysis may be used to recover keystrokes. Song et al. highlighted that the keystroke timing data measured in older SSH implementations [32] may be used to recover encrypted passwords. A risk of compromising emission from keyboards has been postulated by Kuhn and Anderson [20, 17, 2]. They also proposed countermeasures (see US patent [3]). Some unofficial documents on *TEMPEST* [37] often designate keyboards as potential information leaking devices. However, we did not find any experiment or evidence proving or refuting the practical feasibility to remotely eavesdrop keystrokes, especially on modern keyboards.

1.2 Our Contribution

This paper makes the following main contributions:

A Full Spectrum Acquisition Method. To detect compromising electromagnetic emanations a receiver tuned on a specific frequency is generally used. It brings the signal in base band with a limited bandwidth. Therefore, the signal can be demodulated in amplitude (AM) or frequency (FM). This method might not be optimal. Indeed, the signal does not contain the maximal entropy since a significant amount of information is lost. We propose another approach. We acquire the raw signal directly from the antenna and analyze the entire captured electromagnetic spectrum with Short Time Fourier Transform (also known as *Waterfall*) to distill potential compromising emanations.

The Study of Four Different Sources of Information Leakage from Keyboards. To determine if keyboards generate compromising emanations, we measured the electromagnetic radiations emitted when a key is pressed. Due to our improved acquisition method, we discovered several direct and indirect compromising emanations which leak information on the keystrokes. The first technique looks at the emanations of the falling edges (i.e. the transition from a high logic state to a low logic state) from the bi-directional serial cable used in the PS/2 protocol. It can be used to reveal keystrokes with about 1 bit of uncertainty. The second approach uses the same source, but consider the rising and the falling edges of the signal to recover the keystrokes with 0 bits of uncertainty. The third approach is focused on the harmonics emitted by the keyboard to recover the keystrokes with 0 bits of uncertainty. The last approach considers the emanations emitted from the matrix scan routine (used by PS/2, USB and Wireless keyboards) and yields about 2.5 bits of uncertainty per keystroke. This compromising emanation has been previously posited by Kuhn and Anderson [3], although that work provided no detailed analysis.

The Implementation and the Analysis of Four Keystroke Recovery Techniques in Four Different Scenarios. We tested 12 different keyboard models, with PS/2, USB connectors and wireless communication in different setups: a semi-anechoic chamber, a small office, an adjacent office and a flat in a building. We demonstrate that these keyboards are all vulnerable to at least one of the four keystroke recovery techniques in all scenarios. The best attack successfully recovers 95% of the keystrokes at a distance up to 20 meters, even through walls. Because each keyboard has a specific *fingerprint* based on the clock frequency inconsistencies, we can determine the source keyboard of a compromising emanation, even if multiple keyboards from the same model are used at the same time. First, we did the measurements in a semi-anechoic electromagnetic chamber to isolate the device from external noise. Then we confirmed that these compromising emanations are exploitable in real situations.

We conclude that most of modern computer keyboards generate compromising emanations (mainly because of the manufacturer cost pressures in the design). Hence they are not safe to transmit confidential information.

1.3 Structure of the Paper

Section 2 describes some basics on compromising electromagnetic emanations. In Section 3 we present our acquisition method based on Short Time Fourier Transform. In Section 4 we present four different setups used for the measurements, from a semi-anechoic chamber to

real environments. In Section 5 we give the complete procedure used to detect the compromising electromagnetic emanations. Then, we detail the four different techniques. In Section 6, we give the results of our measurements in different setups. In Section 7, we describe some countermeasures to avoid these attacks. In Section 8, we give some extensions and improvements. Finally we conclude.

2 Electromagnetic Emanations

Electromagnetic compatibility (EMC) is the analysis of electromagnetic interferences (EMI) or Radio Frequency Interferences (RFI) related to electric devices. EMC aims at reducing unintentional generation, propagation and reception of electromagnetic energy in electric systems. EMC defines two kinds of unwanted emissions: conductive coupling and radiative coupling. Conductive coupling requires physical support such as electric wires to transmit interferences through the system. Radiative coupling occurs when a part of the internal circuit acts as an antenna and transmits undesired electromagnetic waves. EMC generally distinguishes two types of electromagnetic emissions depending on the kind of the radiation source: differential-mode and common-mode.

Differential-mode radiation is generated by loops formed by components, printed circuit traces, ribbon cables, etc. These loops act as small circular antennas and eventually radiate. These radiations are generally low and do not disturb the whole system. Differential-mode signals are not easily influenced by external radiations. Moreover they can be easily avoided by shielding the system.

Common-mode radiation is the result of undesired internal voltage drops in the circuit which usually appear in the ground loop. Indeed, ground loop currents are due to the unbalanced nature of ordinary transmitting and receiving circuits. Thus, external cables included in the ground loop act as antennas excited by some internal voltages. Because these voltage drops are not intentionally created by the system, it is generally harder to detect and control common-mode radiations than differential-mode radiations.

From the attacker's point of view there are two types of compromising emanations: direct and indirect emanations.

Direct Emanations. In digital devices, data is encoded with logic states, generally described by short burst of square waves with sharp rising and falling edges. During the transition time between two states, electromagnetic waves are eventually emitted at a maximum frequency related to the duration of the rise/fall time. Because these compromising radiations are provided straight by the wire transmitting sensitive data, they are called direct emanations.

Indirect Emanations. Electromagnetic emanations may interact with active electronic components which induce new types of radiations. These unintended emanations manifest themselves as modulations or inter-modulations (phase, amplitude or frequency) or as carrier signals e.g. clock and its harmonics. Non-linear coupling between carrier signals and sensitive data signals, crosstalk, ground pollution or power supply DC pollution may generate compromising modulated signals. These indirect emanations may have better propagation than direct emanations. Hence, they may be captured at a larger range. The prediction of these emanations is extremely difficult. They are generally discovered during compliance tests such as FCC [15], CISPR [10], MIL-STD-461 [22], NACSIM-5000 [37], etc.

3 Electromagnetic Signal Acquisition

Two techniques are generally used to discover compromising electromagnetic emanations.

3.1 Standard Techniques

A method consists in using a spectral analyzer to detect signal carriers. Such a signal can be caught only if the duration of the carrier is significant. This makes compromising emanations composed of peaks difficult to detect with spectral analyzers.

Another method is based on a wide-band receiver tuned on a specific frequency. Signal detection process consists in scanning the whole frequency range of the receiver and to demodulate the signal according to its amplitude modulation (AM) or frequency modulation (FM). When an interesting frequency is discovered, narrow-band antennas and some filters are used to improve the Signal-to-Noise Ratio (SNR) of the compromising emanations. In practice, wide-band receivers such as R-1250 [19] and R-1550 [12] from Dynamic Sciences International, Inc. are used, see [17, 1]. Indeed, these receivers are compliant with secret requirements for the NACSIM-5000 [37] also known as *TEMPEST*. These devices are quite expensive and unfortunately not owned by our lab. Hence, we used a cheaper and open-source solution based on the USRP (Universal Software Radio Peripheral) [14] and the GNU Radio project [35]. The USRP is a device which allows you to create a software radio using any computer with USB port. With different daughterboards, the USRP is able to scan from DC to 2.9 GHz with a sample rate of 64 MS/s at a resolution of 12 bits. The full range on the ADC is 2 volts peak to peak and the input is 50 ohms differential. The GNU

Radio project is a powerful software library used by the USRP to process various modulations (AM, FM, PSK, FSK, etc.) and signal processing constructs (optimized filters, FFT, etc.). Thus, the USRP and the GNU Radio project may act as a wide-band receiver and a spectral analyzer with software-based FFT computation.

3.2 Novel Techniques

Some direct and indirect electromagnetic emanations may stay undetected with standard techniques, especially if the signal is composed of irregular peaks or erratic frequency carriers. Indeed, spectral analyzers need significantly static carrier signals. Similarly, the scanning process of wide-band receivers is not instantaneous and needs a lot of time to cover the whole frequency range. Moreover the demodulation process may hide some interesting compromising emanations.

In this paper, we use a different method to detect compromising electromagnetic emanations of keyboards. First, we obtain the raw signal directly from the antenna instead of a filtered and demodulated signal with limited bandwidth. Then, we compute the Short Time Fourier Transform (STFT), which gives a 3D signal with time, frequency and amplitude.

Modern analog-to-digital converters (ADC) provide very high sampling rates (Giga samples per second). If we connect an ADC directly to a wide-band antenna, we can import the raw sampled signal to a computer and we can use software radio libraries to instantly highlight potentially compromising emanations. The STFT computation of the raw signal reveals the carriers and the peaks even if they are present only for a short time.

Unfortunately there is no solution to transfer the high amount of data to a computer in real time. The data rate is too high for USB 2.0, IEEE 1394, Gigabit Ethernet or Serial ATA (SATA) interfaces. However, with some smart triggers, we can sample only the (small) interesting part of the signal and we store it in a fast access memory. Oscilloscopes provide triggered analog-to-digital converters with fast memory. We used a Tektronix TDS5104 with 1 Mpt memory and a sample rate of 5 GS/s. It can acquire electromagnetic emanations up to 2.5 GHz according to the Nyquist theorem. Moreover, this oscilloscope has antialiasing filters and supports IEEE 488 General Purpose Interface Bus (GPIB) communications. We developed a tool to define some specific triggers (essentially peak detectors) and to export the acquired data to a computer under GNU/Linux over Ethernet. Thus the signal can be processed with the GNU Radio software library and some powerful tools such as Baudline [29] or the GNU project Octave [13]. The advantage of this method is to process the raw signal, which is directly sampled from the antenna without any demodulation. Moreover, all compromising electromagnetic emanations up to a frequency of 2.5 GHz are captured. Thus, with this technique, we are able to highlight compromising emanations quickly and easily. This solution is ideal for very short data burst transmissions used by computer keyboards.

4 Experimental Setup

The objective of this experiment is to observe the existence of compromising emanations of computer keyboards when a key is pressed. Obviously electromagnetic emanations depend on the environment. We defined four different setups.

Setup 1: The Semi-Anechoic Chamber. We used a professional semi-anechoic chamber (7×7 meters). Our aim was not to cancel signal echos but to avoid external electromagnetic pollution (Faraday cage). The antenna was placed up to 5 meters from the keyboard connected to a computer (the maximum distance according to the echo isolation of the room). The tested keyboard was on a one meter high table and the computer (PC tower) was on the ground.

Setup 2: The Office. To give evidence of the feasibility of the attacks with background noise, we measured the compromising emanations of the keyboards in a small office (3×5 meters) with two powered computers and three LCD displays. The electromagnetic background noise was quite important with a cluster of 40 computers 10 meters away from the office, more than 60 powered computers on the same floor and a 802.11n wireless router at less than 3 meters away from the office. The antenna was in the office and moved back through the opened door up to 10 meters away from the keyboard in order to determine the maximum range.

Setup 3: The Adjacent Office. This setup is similar to the office setup but we measured the compromising emanations of the keyboards from an adjacent office through a wall of 15 cm composed of wood and plaster.

Setup 4: The Building. This setup takes place in a flat which is in a building of five floors in the center of a mid-size city. The keyboard was in the fifth floor. We performed measurements with the antenna placed on the same floor first. Then, we moved the antenna as far as the basement (up to 20 meter from the keyboard).

Antennas. Since the compromising emanations were found on frequency bands between 25 MHz and 300 MHz, we used a biconical antenna (50 Ohms VHA 9103 Dipol Balun) to improve the Signal-to-Noise Ratio (SNR). We also tested if these compromising emanations can be captured with a smaller antenna such as a simple loop made of a wire of copper (one meter long).

Keyboards. We picked 12 different keyboard models present in our lab: 7 PS/2 keyboards (Keyboard A1-A7), 2 USB keyboards (Keyboard B1-B2), 2 Laptop keyboards (Keyboard C1-C2) and 1 wireless keyboard (Keyboard D1). They were all bought between 2001 and 2008. We also collected measurements with the keyboard connected to a laptop with battery to avoid possible conductive coupling through the power supply. For obvious security reasons, we do not give the brand name and the model of the tested keyboards.

5 Discovering and Exploiting Emanations

To discover compromising emanations, we placed Keyboard A1 in the semi-anechoic chamber and we used the biconical antenna. A diagram of the experiment is depicted in Figure 1. We acquired the raw signal with the oscilloscope as explained above. Since the memory of the oscilloscope is limited, we have to precisely trigger data acquisition. First, we used the earliest falling edge of the data signal sent when a key is pressed . We physically connected a probe on the data wire of the cable between the keyboard and the computer.

Figure 2 gives the STFT of the captured raw signal when the key E is pressed on an American keyboard. With only one capture we are able to represent the entire spectrum along the full acquisition time. In addition, we have a visual description of all electromagnetic emanations. In particular we clearly see some carriers (vertical lines) and broadband impulses (horizontal lines). The three first techniques are based on these compromising emanations and are detailed in the following sections.

Our objective is to use an electromagnetic trigger, since we normally do not have access to the data wire. The discovered broadband impulses (horizontal lines) can be used as a trigger. Thus, with only an antenna, we are able to trigger the acquisition of the compromising electromagnetic emanations. More details are given below.

Some keyboards do not emit electromagnetic emanations when a key is pressed. But with a different trigger model, based on peak detector as well, we discovered another kind of emission, continuously generated (even if no key is pressed). This is the last technique, detailed in Section 5.4.

5.1 The Falling Edge Transition Technique

To understand how direct compromising emanations may be generated by keyboards, we need to briefly describe the PS/2 communication protocol. According to [9], when a key is pressed, released or held down, the keyboard sends a packet of information known as a *scan code* to the computer. In the default scan code set[1], most of the keys are one-byte long encoded. Some extended keys are two or more bytes long. These codes can be identified by the fact that their first byte is 0xE0. The protocol used to transmit these scan codes is a bi-directional serial communication, based on four wires: Vcc (5 volts), ground, data and clock. For each byte of the scan code, the keyboard pulls down the clock signal at a frequency between 10 KHz and 16.7 KHz for 11 clock cycles. When the clock is low, the state of the signal data is read by the computer. The 11 bits sent correspond to a start bit (0), 8 bits for the scan code of the pressed key (least significant bit first), an odd parity check bit on the byte of the scan code (the bit is set if there is an even number of 1's), and finally a stop bit (1). Figure 3 represents both data and clock signals when the key E is pressed. Note that the scan code is binded to

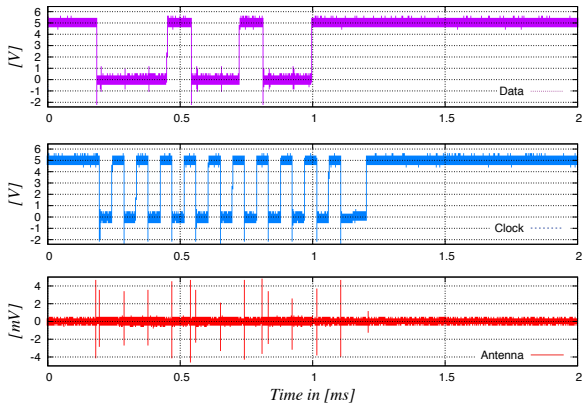

Figure 3: Data, clock and the compromising emanation captured (semi-anechoic chamber, Keyboard A1) with the loop antenna at 5 meters (a wire of copper, one meter long) when the key E (0x24) is pressed. Data signal sends the message: 0 00100100 1 1.

a physical button on the keyboard, it does not represent the character printed on that key. For instance, the scan code of E is 0x24 if we consider the American layout keyboard.

Logic states given by data and clock signals in the keyboard are usually generated by an open collector coupled to a pull-up resistor. The particularity of this system is that the duration of the rising edge is significantly longer (2 μs) than the duration of the falling edge (200 ns). Thus, the compromising emanation of a falling edge should be much more powerful (and with a higher maximum frequency) than the rising edge. This property is known and has been already noticed by Kuhn [17, p.35]. Clock and data signals are identically generated. Hence,

[1] There are three different scan code sets, but the second one is commonly used by default.

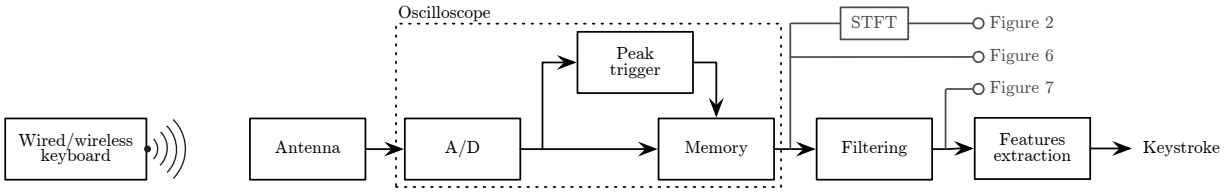

Figure 1: Diagram of our equipment for the experiments.

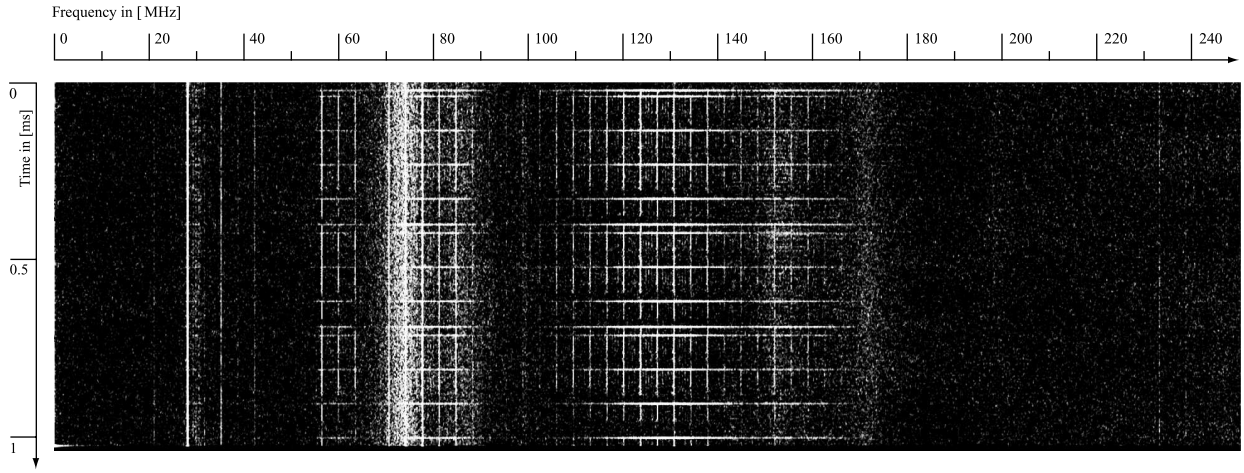

Figure 2: Short Time Fourier Transform (STFT) of the raw signal depicted in Figure 6 (Kaiser windowing of 40, 65536 points)

the compromising emanation detected is the combination of both signals. However (see Figure 3), the edges of the data and the clock lines are not superposed. Thus, they can be easily separated to obtain independent signals.

Since the falling edges of clock signal will always be at the same place, contrary to the falling edges of data signal, we can use them to improve our trigger model. Indeed we consider the detection of a signal based on 11 equidistant falling edges.

Indirect Emanations. If we compare the data signal and the compromising emanation (see Figure 4) we clearly see that the electromagnetic signal is not directly related to the falling edge, as described by Smulders. Indeed, the durations are not equivalent. Thus, the peaks acquired by our antenna seem to be indirectly generated by the falling edges of the combination of clock and data signals. They are probably generated by a peak of current when the transistor is switched. Nevertheless, these emanations, represented by 14 peaks, 11 for the clock signal and 3 for the data signal (see the horizontal lines in Figure 2 or the peaks in Figure 3) partially describe the logic state of the data signal and can be exploited.

Collisions. Because only the falling edges are detected, eventually collisions occur during the keystroke recovery process. For instance, both E (0x24) and G (0x34)

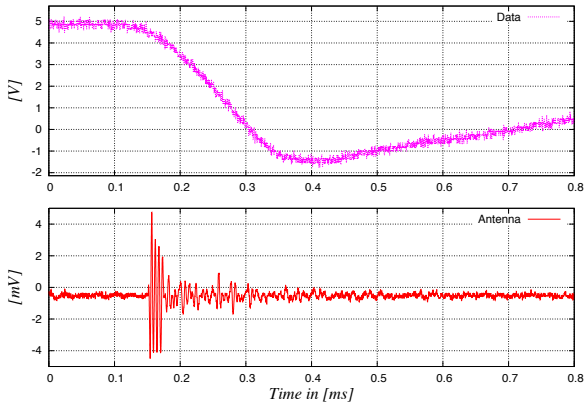

Figure 4: A falling edge of the data signal (upper graph) and the electromagnetic emanation of the keyboard (lower graph). The compromising emission is not directly generated by the data signal such as described by Smulders in [30].

share the same trace if we consider only falling edges. We define the falling edge trace as '2' when both data and clock peaks are detected and '1' when only a clock peak is captured. The letters E (see lower graph in Figure 3) and G may be described by the string `21112112111`.

In Figure 5 we grouped every one byte-long scan code, according to their shared falling edge-based traces.

Trace	Possible Keys
21111111111	<non-US-1>
21111111121	<Release key>
21111111211	F11 KP KP0 SL
21111112111	8 u
21111121111	2 a
21111121211	Caps_Lock
21111211111	F4 `
21111211211	- ; KP7
21111212111	5 t
21112111111	F12 F2 F3
21112111121	Alt+SysRq
21112111211	9 Bksp Esc KP6 NL o
21112112111	3 6 e g
21112121111	1 CTRL_L
21112121211	[
21121111111	F5 F7
21121111211	KP- KP2 KP3 KP5 i k
21121112111	b d h j m x
21121121111	SHIFT_L s y
21121121211	' ENTER]
21121211111	F6 F8
21121211211	/ KP4 l
21121212111	f v
21211111111	F9
21211111211	, KP+ KP. KP9
21211112111	7 c n
21211121111	Alt_L w
21211121211	SHIFT_R \
21211211111	F10 Tab
21211211211	. KP1 p
21211212111	Space r
21212111111	F1
21212111211	0 KP8
21212112111	4 y
21212121111	q
21212121211	=

Figure 5: The one byte-long scan codes classification, according to the falling edges trace for an American keyboard layout.

Even if collisions appear, falling edge traces may be used to reduce the subset of possible transmitted scan codes. Indeed, the average number of potential characters for a falling edge trace is 2.4222 (2.0556 if we consider only alpha-numeric characters and a uniform distribution). For example, an attacker who captured the falling edge-based trace of the word `password` obtains a subset of $3 \cdot 2 \cdot 3 \cdot 3 \cdot 2 \cdot 6 \cdot 2 \cdot 6 = 7776$ potential words, according to Figure 5. Thus, if the objective of the attacker is to recover a secret password, he has significantly reduced the test space (the initial set of $36^8 \approx 2^{41}$ is lowered to 2^{13}). Moreover, if the eavesdropped information concerns an e-mail or a text in English, the plaintext recovery process can be improved by selecting only words contained in a dictionary.

Feature Extraction. The recovery procedure is firstly based on a trigger model, able to detect 11 equidistant peaks transmitted in less than 1 ms. Then, we compute the number of peaks, using a peak-detection algorithm and the GNU Radio library. The feature extraction is based on the number of peaks correlated to the most probable value of the table depicted in Figure 5. The main limitation of the recovery procedure is the ability to trigger this kind of signal.

5.2 The Generalized Transition Technique

The previously described attack is limited to a partial recovery of the keystrokes. This is a significant limitation. We know that between two '2' traces, there is exactly one data rising edge. If we are able to detect this transition we can fully recover the keystrokes.

To highlight potential compromising emanations on the data rising edge, we use a software band-pass filter to isolate the frequencies of the broadband impulses (e.g. 105 MHz to 165 MHz of the raw signal in Figure 2). Figure 7 corresponds to the filtered version of the raw time-domain signal represented in Figure 6. We remark

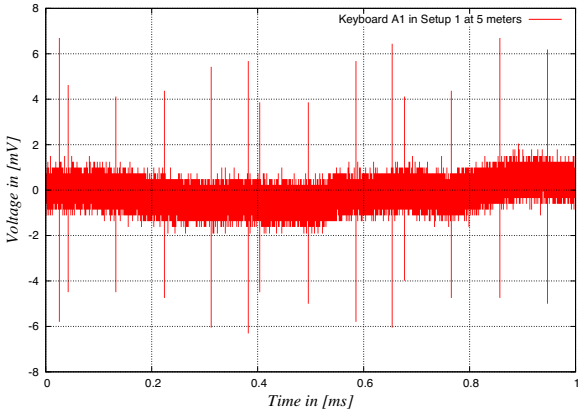

Figure 6: Raw signal (Keyboard A1, Setup 1 at 5 meters with the biconical antenna) when the key E is pressed.

that the filtering process significantly improves the SNR. Thus, the peak detection algorithm is much more efficient.

Furthermore, we notice that the energy of the peaks of the clock falling edges is not constant. Empirically, clock peaks have more energy when the state of data signal is high. Indeed, the data signal pull-up resistor is open. When the clock signal is pulled down, the surplus of energy creates a stronger peak. Hence, the peaks generated by the falling edge of the clock signal intrinsically encode the logic state of the data signal. Because

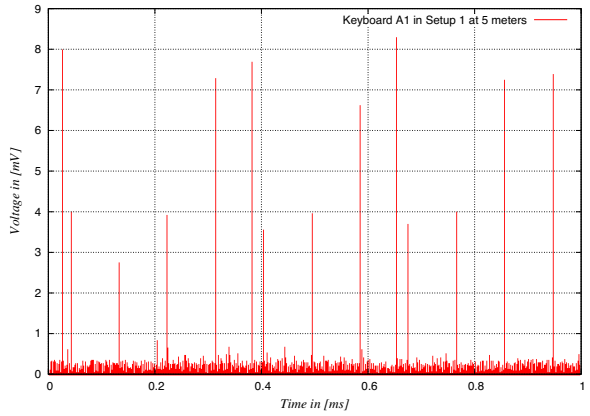

Figure 7: Band-pass (105-165MHz) filtered signal of Figure 6.

there is exactly one rising edge between two falling edge traces of '2', we simply consider the highest clock peak as the rising edge data transition. For example in Figure 7, the rising edge data transitions are respectively at peaks 5 and 9. Thus, the complete data signal is 0010 0100 which corresponds to 0x24 (E). Thus, we manage to completely recover the keystrokes. Note that the band-pass filter improves the previous attack as well. However, the computation cost prevents real time keystroke recovery without hardware accelerated filters.

Feature Extraction. The recovery procedure is firstly based on the same trigger model described previously (11 equidistant peaks detected in less than 1 ms). Then, we filter the signal to consider only the frequency bands containing the peak impulses. The feature extraction is based on the detected peaks. First, we define the threshold between a high peak and a low peak thanks to the two first peaks. Indeed, because we know that data and clock are pulled down, the first one corresponds to a state where clock is high and data is low and the second one describes the state where both signals are low. Then, we determine the potential (and colliding) keystrokes with Figure 5. In our example, it corresponds to the keys 3,6,E,G. Then, we select some bits which differentiate these keys. According to their scan code 3=0x26, 6=0x36, E=0x24, G=0x34 we check the state of the peaks 4 and 8 in Figure 7, which correspond to respectively the second and the fifth bit of the scan codes. Because they are both low, we conclude that the transmitted key is E.

5.3 The Modulation Technique

Figure 2 highlights some carriers with harmonics (vertical lines between 116 MHz and 147 MHz). These compromising electromagnetic emissions come from unintentional emanations such as radiations emitted by the clock, non-linear elements, crosstalk, ground pollution, etc. Determining theoretically the reasons of these compromising radiations is a very complex task. Thus, we can only sketch some probable causes. The source of these harmonics corresponds to a carrier of approximately 4 MHz which is very likely the internal clock of the microcontroller inside the keyboard. Interestingly, if we correlate these harmonics with both clock and data signals, we clearly see modulated signals (in amplitude and frequency) which fully describe the state of both clock and data signals, see Figure 8. This means that the scan code can be completely recovered from these harmonics.

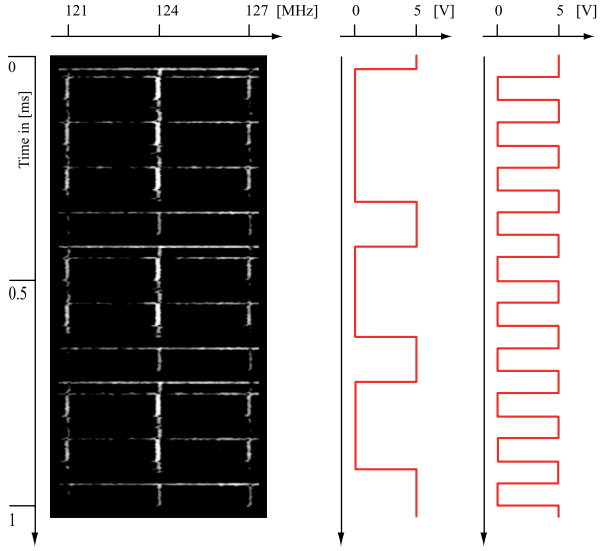

Figure 8: The amplitude and frequency modulations of the harmonic at 124 MHz correlated to both data and clock signals (Keyboard A1, semi-anechoic chamber at 5 meters).

Note that even if some strong electromagnetic interferences emerge, the attacker may choose non-jammed harmonics to obtain a clear signal. It is even possible to superpose them to improve the SNR. Compared to the previous techniques, the carrier-based modulation is much more interesting for distant reception. Indeed, AM and FM transmissions are generally less disrupted by noise and obstacles such as walls, floors, etc. Moreover this technique is able to fully recover the keystrokes. These indirect emanations – which have no formal explanation, but are probably based on crosstalk with the ground, the internal clock of the microcontroller, data and clock signals – let the attacker recover the keystrokes of a keyboard.

This experiment shows that cheap devices such as keyboards may radiate indirect emanations, which are much

more compromising than direct emanations. Even if the SNR is smaller, the use of a frequency modulation significantly improves the eavesdropping range. Moreover, the attacker may avoid some noisy frequency bands by selecting only the clearest harmonics. Furthermore, indirect emanations completely describe both clock and data signals.

Feature Extraction. The feature extraction is based on the demodulation in frequency and amplitude of the captured signal centered on the strongest harmonic. In our example and according to Figure 8 the carrier corresponds to 124 MHz. We used the GNU Radio library to demodulate the signal. However, we still need to use the trigger model based on peak detector since the memory of the oscilloscope is limited. Another option is to directly capture the signal with the USRP. Indeed, the lower but continuous sampling rate of the USRP is sufficient to recover the keystrokes. Unfortunately, the sensitivity of the USRP is weaker than the oscilloscope and the eavesdropping range is limited to less than 2 meters in the semi-anechoic chamber.

5.4 The Matrix Scan Technique

The techniques described above are related to the use of PS/2 and some laptop keyboards. However, new keyboards tend to use USB or wireless communication. In this section, we present another compromising emanation which concerns all keyboard types: PS/2, USB, Notebooks and even wireless keyboards. This attack was previously postulated by Kuhn and Anderson [20] but no practical data has appeared so far in the open literature.

Almost all keyboards share the same pressed key detection routine. A major technical constraint is to consider a key as pressed if the button is pushed for 10 ms, see US Patent [31]. Thus every pressed key should be detected within this time delay. From the manufacturer's point of view, there is another main constraint: the cost of the device. A naive solution to detect pressed keys is to poll each key in a row. This solution is clearly not optimal since it requires a large scan loop routine and thus longer delays. Moreover important leads (i.e. one circuit for each key) increase the cost of the device.

A smart solution [31] is to arrange the keys in a *matrix*. The keyboard controller, often a 8-bit processor, parses columns one-by-one and recovers the state of 8 keys at once. This *matrix scan* process can be described as 192 keys (some keys may not be used, for instance modern keyboards use 104/105 keys) arranged in 24 columns and 8 rows. Columns are connected to a driver chip while rows are connected to a detector chip. Keys are placed at the intersection of columns and rows. Each key is an analog switch between a column and a row. The keyboard controller pulses each column through the driver (using the address bus and the strobe signal). The detector measures the states of the 8 rows. Note that a row is connected to 24 keys, but only one may be active, the one selected by the driver. Suppose we pressed the key corresponding to column 3 and row 5. The controller pulses columns . . ., 22, 23, 24, 1, 2 with no key event. Now, the controller pulses column 3. Row 5, which corresponds to the pressed key, is detected. The keyboard starts a subroutine to transmit the scan code of the key to the computer. This subroutine takes some time. Thus, the next column pulse sent by the scan routine is delayed.

Columns in the matrix are long leads since they connect generally 8 keys. According to [31], these columns are continuously pulsed one-by-one for at least 3μs. Thus, these leads may act as an antenna and generate electromagnetic emanations. If an attacker is able to capture these emanations, he can easily recover the column of the pressed key. Indeed, the following pulse will be delayed.

To figure out if these emanations can be captured, we picked Keyboard A6 and acquired the signal being one meter from the keyboard in the semi-anechoic chamber with a simple one meter long wire of copper as antenna. Figure 9 gives the repeated peak burst continuously emitted by the keyboard. Figure 10 shows the zoomed compromising emanations when the key C resp. key H is pressed.

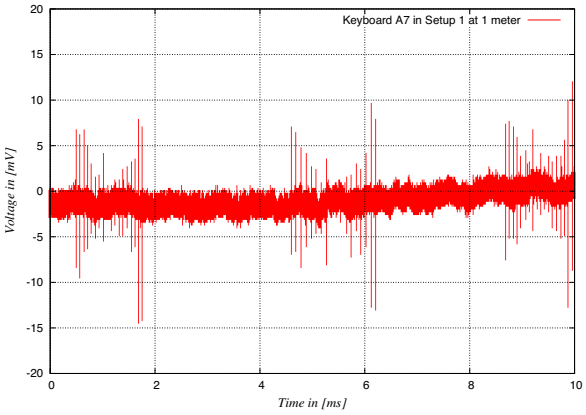

Figure 9: A large view of compromising emanations exploited by the Matrix Scan Technique, (Keyboard A7, semi-anechoic chamber at 1 meter).

The key matrix arrangement may vary, depending on the manufacturer and the keyboard model. We dismantled a keyboard and analyzed the key circuit layout to retrieve the matrix key specifications. The right part of the keyboard layout is depicted on Figure 11. We clearly identify a column (black) and fours rows.

Figure 12 represents the groups of alphanumeric scan codes according to their indirect compromising emana-

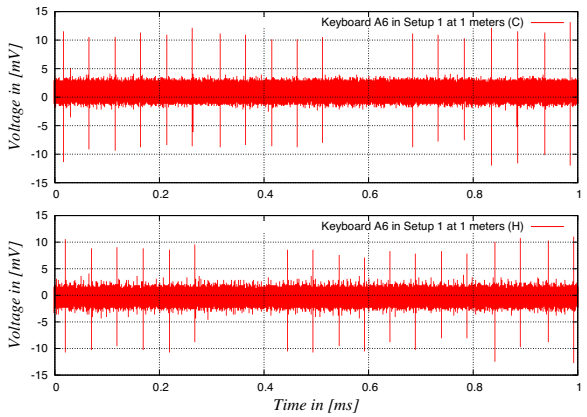

Figure 10: The matrix scan emanations for the letters C and H (Keyboard A6, Setup 1 at 1 meter).

tions (or column number) for Keyboard A6. We describe each electromagnetic signal as a number corresponding to the delayed peak. For example, in Figure 10, the key C is described as 12 and the key H as 7.

Even if this signal does not fully describe the pressed key, it still gives partial information on the transmitted scan code, i.e. the column number. So, as described in the Falling Edge Transition Technique, collisions occurs between key codes. Note that this attack is less efficient than the first one since it has (for this specific keyboard) in average 5.14286 potential key codes for a keystroke (alpha-numeric only). However, an exhaustive search on the subset is still a major improvement.

Note that the matrix scan routine loops continuously. When no key is pressed, we still have a signal composed of multiple equidistant peaks. These emanations may be used to remotely detect the presence of powered computers.

Concerning wireless keyboards, the wireless data burst transmission can be used as an electromagnetic trigger to detect exactly when a key is pressed, while the matrix scan emanations are used to determine the column it belongs to. Moreover the ground between the keyboard and the computer is obviously not shared, thus the compromising electromagnetic emanations are stronger than those emitted by wired keyboards. Note that we do not consider the security of the wireless communication protocol. Some wireless keyboards use a weakly or not encrypted channel to communicate with the computer, see [8, 23].

Feature Extraction. To partially recover keystrokes, we continuously monitor the compromising emanations of the matrix scan routine with a specific trigger model. According to Figure 12 the six first peaks are always present, as well as the last three peaks. Indeed, these peaks are never missing (or delayed). Thus, we use this

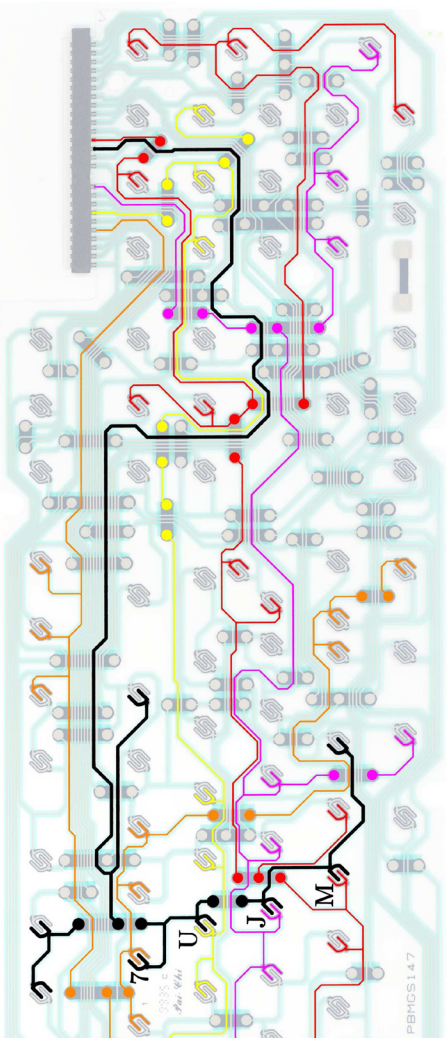

Figure 11: Scan matrix polls columns one-by-one. We are able to deduce on which column the pressed key belongs to. On this keyboard, there will be a collision between keystrokes 7, U, J, M, and others non alpha-numeric keys such as F6, F7, ^, and the dot.

Peak trace	Possible Keys
7	6 7 h J M N U Y
8	4 5 B F G R T V
9	Backspace ENTER
10	9 L O
11	0 P
12	3 8 C D E I K
13	1 2 S W X Z
14	SPACE A Q

Figure 12: The alpha-numeric key classification according to the key scanning routing compromising emanations (Keyboard A6 with American layout).

fixed pattern to define a trigger model. Moreover, the matrix scan continuously radiates compromising emanations since the key is pressed. When a keystroke subset is detected, we acquire multiple samples until another pattern is detected. Therefore, we pick the most often captured pattern.

5.5 Distinguishing Keystrokes from Multiple Keyboards

The falling edge-based traces are distinguishable depending on the keyboard model. Indeed, according to the frequency of the peaks, the clock frequency inconsistencies, the duration between clock and data falling edges, we are able to deduce a specific *fingerprint* for every keyboard. When multiple keyboards are radiating at the same time, we are able to identify and differentiate them. For example, we measured a clock frequency of 12.751 KHz when a key was pressed on a keyboard and the clock frequency was 13.752 KHz when a key was pressed on another keyboard. Thus, when an emanation is captured, we measure the time between two falling edges of the clock and then we deduce if the scan code comes from first or the second keyboard. In practice, we were able to differentiate all the keyboards we tested, even if the brand and the model were equivalent.

This method can be applied to the Falling Edge Transition Technique, the Generalized Transition Technique and the Modulation Technique since they rely on the same kind of signal. The distinguishing process for the Modulation Technique can even be improved by using the clock frequency inconsistencies of the microcontroller as another identifier. For the Matrix Scan Technique, the compromising electromagnetic emanation burst emitted every 2.5 ms (see Figure 9) can be used as a synchronization signal to identify a specific keyboard emission among multiple keyboards. Additionally, the duration between the scan peaks is different, depending on the keyboard model. Thus, it may be used to identify the source keyboard. However, the continuous emission significantly deteriorates the identification process.

Another physical element can be used to distinguish keystrokes from multiple keyboards. For the three first techniques, the broadband impulse range is determined by the length of the keyboard cable, which forms a resonant dipole. Thus, we can use this particularity to identify the source of a compromising emanation. An interesting remark is that the length of the wire connecting the computer to the keyboard is shorter in notebooks. The frequency band of the compromising emanation is higher and the SNR smaller. The Matrix Scan Technique emanates at a higher frequency since the leads of the keyboard layout, acting as an antenna, are shorter.

6 Evaluation in Different Environments

While we have demonstrated techniques that should be able to extract information from keyboard emanations, we have not studied how they are affected by different environments. In this section we study the accuracy of our approaches in all the environments described. Our analysis indicates that keyboard emanations are indeed problematic in practical scenarios.

Evaluating the emission risks of these attacks is not an easy task. Indeed, these results highly depend on the antenna, the trigger model, pass-band filters, peak detection, etc. Moreover, we used trivial filtering processes and basic signal processing techniques. These methods could be significantly improved using beamforming, smart antennas, better filters and complex triggers. In addition, measurements in real environments but the semi-anechoic chamber were subject to massive change, depending on the electromagnetic interferences. Figure 13 gives the list of vulnerable keyboards in all setups, according to the four techniques previously described. Note that all the tested keyboards (PS/2, USB, wireless and laptop) are vulnerable to at least one of these attacks. First, we present the measurements in Setup 1 (semi-anechoic chamber) to guarantee some stable results.

Keyboard	Type	FETT	GTT	MT	MST
A1	PS/2	✓	✓	✓	✓
A2	PS/2	✓	✓		✓
A3	PS/2	✓	✓	✓	✓
A4	PS/2	✓	✓	✓	
A5	PS/2	✓	✓	✓	
A6	PS/2	✓	✓		✓
A7	PS/2	✓			✓
B1	USB				✓
B2	USB				✓
C1	LT	✓	✓		✓
C2	LT				✓
D1	Wi				✓

Figure 13: The vulnerability of the tested keyboards according to the Falling Edge Transition Technique (FETT), the Generalized Transition Technique (GTT), the Modulation Technique (MT) and the Matrix Scan Technique (MST).

6.1 Results in the Semi-Anechoic Chamber

We consider an attack as successful when we are able to correctly recover more than 95% of more than 500 keystrokes. The Falling Edge Transition Technique, the Generalized Transition Technique and the Modulation Technique are successful in the semi-anechoic chamber

for all vulnerable keyboards. This means that we can recover the keystrokes (fully or partially) to at least 5 meters (the maximum distance inside the semi-anechoic chamber). However, the Matrix Scan Technique is limited to a range of 2 to 5 meters, depending on the keyboard. Figure 14 represents the probability of success of the Matrix Scan Technique according to the distance between the tested keyboard and the antenna.

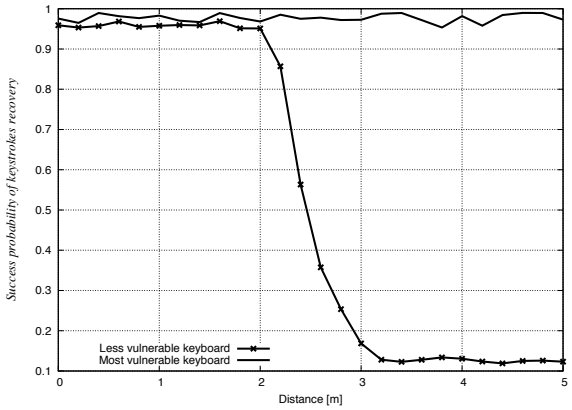

Figure 14: The success probability of the Matrix Scan Technique in the semi-anechoic chamber according to the distance.

We notice that the transition between a successful and a missed attack is fast. Indeed, The correctness of the recovery process is based on the trigger of the oscilloscope. If a peak is not detected, the captured signal is incomplete and the recovered keystroke is wrong. Thus, under a SNR of 6 dB there is nearly no chance to successfully detect the peaks. The SNR is computed according to the average value of the peaks in volts divided by the RMS of the noise in volts.

Considering 6 dB of SNR as a minimum, we are able to estimate the theoretical maximum distance to successfully recover the keystrokes for all techniques in the semi-anechoic chamber. Figure 15 gives the estimated maximum distance range according to the weakest and the strongest keyboard.

In Figure 16 the upper graph gives the SNR of the Falling Edge Transition Technique and the Generalized Transition Technique on Keyboard A1 from 1 meter to 5 meters. The middle graph details the SNR (in dB) of the strongest frequency carrier of the Modulation Technique for the same keyboard. Thus, we can estimate the maximum range of these attacks according to their SNR. The lower graph gives the SNR of the Matrix Scan Technique for the same keyboard. All the measurements were collected in the semi-anechoic chamber.

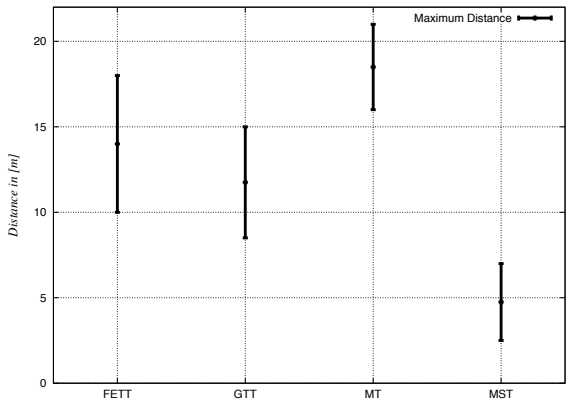

Figure 15: The theoretically estimated maximum distance range to successfully recover 95% of the keystroke according the four techniques in the semi-anechoic chamber, from the less vulnerable to the most vulnerable keyboard.

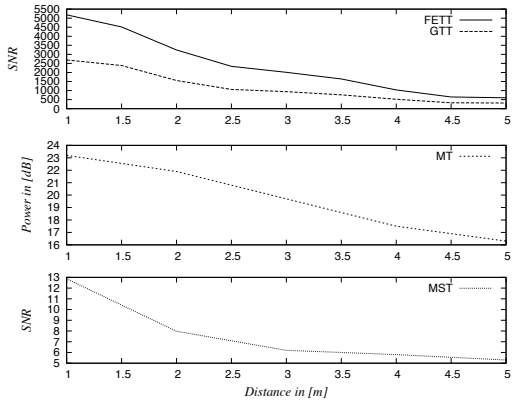

Figure 16: Signal-to-Noise ratio of the peaks [V] / RMS of the noise [V] for the Falling Edge Transition Technique and the Generalized Transition Technique (upper graph). SNR [dB] of the compromising carrier of the Modulation Technique (middle graph). SNR of the peaks [V] / RMS of the noise [V] for Matrix Scan Technique (lower graph).

6.2 Results in Practical Environments

The second phase is to test these techniques in some practical environments. The main difference is the presence of a strong electromagnetic background noise. However, all the techniques remain applicable.

Setup 2: The Office. Figure 17 gives the probability of success of the Generalized Transition Technique on Keyboard A1 measured in the office according to the distance between the antenna and the keyboard. We notice that the sharp transition is present as well when the SNR of the peaks falls under 6 dB. The maximum range of this at-

tack is between 3 and 7.5 meters depending on the tested keyboard. Note that these values were unstable due to a changing background noise. They correspond to an average on multiple measurements.

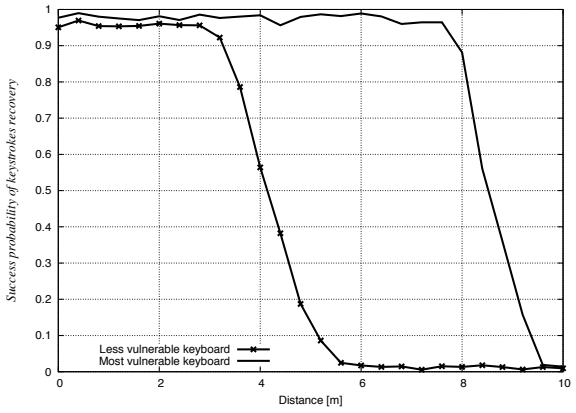

Figure 17: The success probability of the Generalized Transition Technique in the office, according to the distance between the keyboard and the antenna (biconical).

The Modulation Technique is based on a signal carrier. The SNR of this carrier should determine the range of the attack. However, we obtained better results with the same trigger model used in the Falling Edge Transition Technique and the Generalized Transition Technique than one based on the carrier signal only.

Because the Matrix Scan Technique is related to the detection of the peaks, we noticed the same attenuation when the SNR falls under 6 dB. Figure 18 gives the maximum range for the four techniques measured in the office.

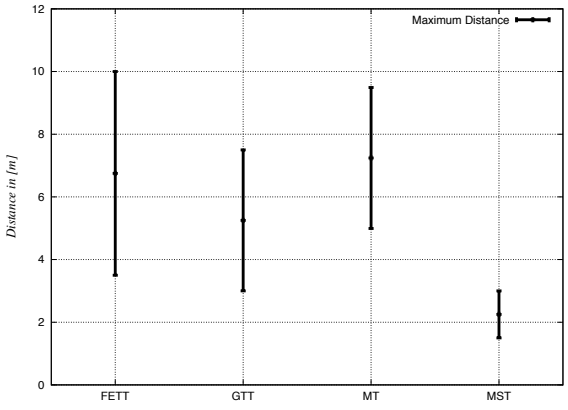

Figure 18: Maximum distance ranges, from the least vulnerable keyboard to the most vulnerable keyboard, to successfully recover 95% of the keystroke according to the techniques (in the office with the biconical antenna).

Setup 3: The Adjacent Office. Results on this setup are basically the same as the previous setup (the office), except that the wall made of plaster and wood removes 3 dB to the SNR.

Setup 4: The Building. We notice some unexpected results in this setup. Indeed, we are able to capture the signal and successfully recover the keystroke with a probability higher than 95% 20 meters away from the keyboard (i.e. the largest distance inside the building). Sometimes the environment can be extremely favorable to the eavesdropping process. For example, metallic structures such as pipes or electric wires may act as antennas and significantly improve the eavesdropping range. In this case, the compromising emanations are carried by the shared ground of the electric line. Thus, the range is defined by the distance between the keyboard and the shared ground and the distance between the shared ground and the antenna. Note that the Matrix Scan Technique is easily disrupted by a noisy shared ground, since the trigger model is more complicated and the emanations weaker. For this technique, we were only able to successfully capture compromising emanations when the keyboard is at less than one meter away from the shared ground. This setup is interesting because it corresponds to a practical scenario where the eavesdropper is placed in the basement of a building and tries to recover the keystrokes of a keyboard at the fifth floor. Unfortunately, it was impossible to provide stable measurements since they highly depend on the environment. We noticed that the main (metallic) water pipe of the building acts as an antenna as well and can be used in place of the shared ground. Furthermore, this *antenna* is less polluted by electronic devices.

Perfect Trigger. We tried the same experiment in the office, but the background noise was too strong. Indeed, we were not able to successfully detect the compromising emissions. However, with a probe physically connected to the data wire, we correctly triggered the emanations. Indeed, the electromagnetic compromising emissions are present in the shared ground. The limitation concerns only the trigger. All the techniques were applicable on the whole floor (about 20 meters) with the keyboard one meter away from the shared ground.

Obviously, you can directly connect the oscilloscope to the shared ground of the building to eavesdrop the keystrokes. Note that an old PC tower used to supply tested keyboards carries the compromising emanations directly through the shared ground. But, this is out of the scope of this paper since we focused our research on electromagnetic emanations only. To avoid such conductive coupling through power supply, we performed our measurements with the keyboards connected to a battery powered laptop.

7 Countermeasures

In this Section, we suggest some possible countermeasures to protect keyboards against the four attacks.

The first solution to avoid the compromising emanations seems trivial. We should shield the keyboard to significantly reduce all electromagnetic radiations. Many elements inside the keyboard may generate emanations: the internal electronic components of the keyboard, the communication cable, and the components of the motherboard inside the computer. Thus, to eliminate these emanations, we have to shield the whole keyboard, the cable, and a part of the motherboard of the computer. We discussed with a manufacturer and he pointed out that the price to shield the entire keyboard will at least double the price of the device. This solution may not be applicable for cost reasons. One can find on the market some keyboards which respect the NATO SDIP-27 standard. All these documents remain classified and no information is available on the actual emission limit or detailed measurement procedures. Another solution is to protect the room where vulnerable keyboards are used. For example, the room can be shielded or a secure physical perimeter can be defined around the room, for instance 100 meters. Attacks 1, 2 and 3 are directly related to the PS/2 protocol. One solution to avoid unintended information leaks is to encrypt the bi-directional serial communication, see [3]. In modern keyboards, one chip contains the controller, the driver, the detector, and the communication interface. So, the encryption may be computed in this chip and no direct compromising emanations related to the serial communication will appear. Attack 4 is related to the scan matrix loop. A solution could be to design a new scanning process algorithm. Even if keyboards still use scan matrix loop routine, there exists some applicable solutions. As described by Anderson and Kuhn [3], the loop routine can be randomized. Actually columns are scanned in the incremental order 1, 2, 3, . . ., 23, 24, but it seems possible to change the order randomly. Moreover, we can add some random delays during the scanning loop process to obfuscate the execution of the subroutine. Both solutions do not avoid electromagnetic emanations, but makes the keystrokes recovery process theoretically impossible. Paavilainen [27] also proposed a solution. It consists in high-frequency filtering matrix signals before they are fed into the keyboard. This will significantly limits compromising electromagnetic emanations.

8 Extensions

Our study has shown that electromagnetic emanations of modern wired and wireless keyboards may be exploited from a distance to passively recover keystrokes. In this section, we detail some extensions and remarks.

The main limitation of these attacks concerns the trigger of the data acquisition. This can be improved with an independent process, using specific filters between the antenna and the ADC. Additionally, other compromising emanations such as the sound of the pressed key could be used as trigger. Furthermore, modern techniques such as beamforming could significantly improve the noise filtering.

Another improvement would be to simultaneously leverage multiple techniques. For keyboards that are vulnerable to more than one technique, we could correlate the results of the different techniques to reduce uncertainty in our guesses.

Another extension would be to accelerate these attacks with dedicated hardware. Indeed, the acquisition time (i.e. the transfer of the data to a computer), the filtering and decoding processes take time (about two seconds per keystroke). With dedicated system and hardware-based computation such as FPGAs, the acquisition, filtering and decoding processes can obviously be instantaneous (e.g. less than the minimum time between two keystrokes). However, the keystrokes distinguishing process when multiple keyboards are radiating is still difficult to implement especially for the Matrix Scan Technique, since the acquisition process should be continuous.

We spend time experimenting with different types of antennas and analog-to-digital converters. In particular, we used the USRP and the GNU Radio library to avoid the need of an oscilloscope and to obtain a portable version of the Modulation Technique. Indeed, we can hide the USRP with battery and a laptop in a bag, the antenna can be replaced by a simple wire of copper (one meter long) which is taped on the attacker's body hidden under his clothes. With this transportable setup, we are able to recover keystrokes from vulnerable keyboards stealthily. However the eavesdropping range is less than two meters.

9 Conclusion

We have provided evidence that modern keyboards radiate compromising electromagnetic emanations. The four techniques presented in this paper prove that these inexpensive devices are generally not sufficiently protected against compromising emanations. Additionally, we show that these emanations can be captured with relatively inexpensive equipment and keystrokes are recovered not only in the semi-anechoic chamber but in some practical environments as well.

The consequences of these attacks is that compromising electromagnetic emanations of keyboards still represent a security risk. PS/2, USB laptop and wireless

keyboards are vulnerable. Moreover, there is no software patch to avoid these attacks. We have to replace the hardware to obtain safe devices. Due to cost pressure in the design, manufacturers may not systematically protect keyboards. However, some (expensive) secure keyboards already exist but they are mainly bought by military organizations or governments.

The discovery of these attacks was directly related to our method based on the analysis of the entire spectrum and the computation of Short Time Fourier Transform. This technique has some pros such as the human-based visual detection of compromising emanations, the large spectrum bandwidth, the use of the raw signal without RF front-ends and the post-demodulation using software libraries. The cons are the limited memory and the difficulty to obtain efficient triggers. However, for short data bursts, this solution seems relevant.

Future works should consider similar devices, such as keypads used in cash dispensers (ATM), mobile phone keypads, digicodes, printers, wireless routers etc. Another major point is to avoid the use of a peak detection algorithm since it is the main limitation of these attacks. The algorithms of the feature extractions could be improved as well. The correlation of these attacks with non-electromagnetic compromising emanation attacks such as optical, acoustic or time attacks could significantly improve the keystroke recovery process.

We discussed with a few agencies interested by our videos [36]. They confirmed that this kind of attack has been practically done since the 1980's on old computer keyboards, with sharp transitions and high voltages. However, they were not aware on the feasibility of these attacks on modern keyboards. Some of these attacks were not known to them.

Acknowledgments

We gratefully thank Pierre Zweiacker and Farhad Rachidi from the Power Systems Laboratory (EPFL) for the semi-anechoic chamber and their precious advices. We also thank Eric Augé, Lucas Ballard, David Jilli, Markus Kuhn, Eric Olson and the anonymous reviewers for their extremely constructive suggestions and comments.

References

[1] AGRAWAL, D., ARCHAMBEAULT, B., RAO, J. R., AND ROHATGI, P. The EM Side-Channel(s). In *CHES* (2002), B. S. K. Jr., Çetin Kaya Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer, pp. 29–45.

[2] ANDERSON, R. J., AND KUHN, M. G. Soft Tempest – An Opportunity for NATO. *Protecting NATO Information Systems in the 21st Century, Washington, DC, Oct 25-26* (1999).

[3] ANDERSON, R. J., AND KUHN, M. G. Lost Cost Countermeasures Against Compromising Electromagnetic Computer Emanations. United States Patent US 6,721,324 B1, 2004.

[4] ASONOV, D., AND AGRAWAL, R. Keyboard Acoustic Emanations. In *IEEE Symposium on Security and Privacy* (2004), IEEE Computer Society, pp. 3–11.

[5] BACKES, M., DÜRMUTH, M., AND UNRUH, D. Compromising reflections-or-how to read lcd monitors around the corner. In *IEEE Symposium on Security and Privacy* (2008), P. McDaniel and A. Rubin, Eds., IEEE Computer Society, pp. 158–169.

[6] BALZAROTTI, D., COVA, M., AND VIGNA, G. Clearshot: Eavesdropping on keyboard input from video. In *IEEE Symposium on Security and Privacy* (2008), P. McDaniel and A. Rubin, Eds., IEEE Computer Society, pp. 170–183.

[7] BERGER, Y., WOOL, A., AND YEREDOR, A. Dictionary attacks using keyboard acoustic emanations. In *ACM Conference on Computer and Communications Security* (2006), A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds., ACM, pp. 245–254.

[8] BRANDT, A. Privacy Watch: Wireless Keyboards that Blab, January 2003. http://www.pcworld.com/article/108712/privacy_watch_wireless_keyboards_that_blab.html.

[9] CHAPWESKE, A. The PS/2 Mouse/Keyboard Protocol. http://www.computer-engineering.org/.

[10] CISPR. The International Special Commitee on Radio Interference. http://www.iec.ch/zone/emc/emc_cis.htm.

[11] CORRELL, J. T. Igloo White - Air Force Magazine Online 87, 2004.

[12] DYNAMIC SCIENCES INTERNATIONAL, INC. R-1550a tempest receiver, 2008. http://www.dynamicsciences.com/client/show_product/33.

[13] EATSON, J. GNU Octave, 2008. http://www.gnu.org/software/octave/.

[14] ETTUS, M. The Universal Software Radio Peripheral or USRP, 2008. http://www.ettus.com/.

[15] FCC. Federal Communications Commission. http://www.fcc.gov.

[16] GANDOLFI, K., MOURTEL, C., AND OLIVIER, F. Electromagnetic analysis: Concrete results. In *CHES* (2001), Çetin Kaya Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *Lecture Notes in Computer Science*, Springer, pp. 251–261.

[17] KUHN, M. G. Compromising Emanations: Eavesdropping risks of Computer Displays. *Technical Report UCAM-CL-TR-577* (2003).

[18] KUHN, M. G. Security limits for compromising emanations. In *CHES* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 265–279.

[19] KUHN, M. G. Dynamic Sciences R-1250 Receiver, 2008. http://www.cl.cam.ac.uk/ mgk25/r1250/.

[20] KUHN, M. G., AND ANDERSON, R. J. Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations. In *Information Hiding* (1998), D. Aucsmith, Ed., vol. 1525 of *Lecture Notes in Computer Science*, Springer, pp. 124–142.

[21] LOUGHRY, J., AND UMPHRESS, D. A. Information leakage from optical emanations. *ACM Trans. Inf. Syst. Secur. 5*, 3 (2002), 262–289.

[22] MIL-STD-461. Electromagnetic Interference Characteristics Requirements for Equipment. https://acc.dau.mil/ CommunityBrowser.aspx?id=122817.

[23] MOSER, M., AND SCHRODEL, P. 27MHz Wireless Keyboard Analaysis Report, 2005. http://www.blackhat.com/presentations/bh-dc-08/ Moser/Whitepaper/bh-dc-08-moser-WP.pdf.

[24] MULDER, E. D., ÖRS, S. B., PRENEEL, B., AND VERBAUWHEDE, I. Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems. *Computers & Electrical Engineering 33*, 5-6 (2007), 367–382.

[25] NALTY, B. C. *The war against trucks: aerial interdiction in southern Laos, 1968-1972*. Air Force History and Museums Program, United States Air Force, 2005.

[26] NATIONAL SECURITY AGENCY. TEMPEST: A Signal Problem, 2007. http://www.nsa.gov/public_info/_files/ cryptologic_spectrum/tempest.pdf.

[27] PAAVILAINEN, R. Method and device for signal protection. United States Patent US 7,356,626 B2, 2008.

[28] QUISQUATER, J.-J., AND SAMYDE, D. Electromagnetic analysis (ema): Measures and countermeasures for smart cards. In *E-smart* (2001), I. Attali and T. P. Jensen, Eds., vol. 2140 of *Lecture Notes in Computer Science*, Springer, pp. 200–210.

[29] SIGBLIPS DSP ENGINEERING. Baudline, 2008. http://www.baudline.com.

[30] SMULDERS, P. The Threat of Information Theft by Reception of Electromagnetic Radiation from RS-232 Cables. *Computers and Security 9*, 1 (1990), 53–58.

[31] SONDERMAN, E. L., AND DAVIS, W. Z. Scan-controlled keyboard. United States Patent US 4,277,780, 1981.

[32] SONG, D. X., WAGNER, D., AND TIAN, X. Timing analysis of keystrokes and timing attacks on ssh. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2001), USENIX Association, pp. 25–25.

[33] TANAKA, H. Information leakage via electromagnetic emanations and evaluation of tempest countermeasures. In *ICISS* (2007), P. D. McDaniel and S. K. Gupta, Eds., vol. 4812 of *Lecture Notes in Computer Science*, Springer, pp. 167–179.

[34] VAN ECK, W. Electromagnetic radiation from video display units: an eavesdropping risk? *Comput. Secur. 4*, 4 (1985), 269–286.

[35] VARIOUS AUTHORS. The GNU Software Radio, 2008. http://www.gnuradio.org/.

[36] VUAGNOUX, M., AND PASINI, S. Videos of the Compromising Electromagnetic Emanations of Wired Keyboards, October 2008. http://lasecwww.epfl.ch/keyboard/.

[37] YOUNG, J. NSA Tempest Documents, 2008. http://cryptome.info/0001/nsa-tempest.htm.

[38] ZHUANG, L., ZHOU, F., AND TYGAR, J. D. Keyboard acoustic emanations revisited. In *ACM Conference on Computer and Communications Security* (2005), V. Atluri, C. Meadows, and A. Juels, Eds., ACM, pp. 373–382.

Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems

Kehuan Zhang
Indiana University, Bloomington
kehzhang@indiana.edu

XiaoFeng Wang
Indiana University, Bloomington
xw7@indiana.edu

Abstract

A multi-user system usually involves a large amount of information shared among its users. The security implications of such information can never be underestimated. In this paper, we present a new attack that allows a malicious user to eavesdrop on other users' keystrokes using such information. Our attack takes advantage of the stack information of a process disclosed by its virtual file within *procfs*, the process file system supported by Linux. We show that on a multi-core system, the ESP of a process when it is making system calls can be effectively sampled by a "shadow" program that continuously reads the public statistical information of the process. Such a sampling is shown to be reliable even in the presence of multiple users, when the system is under a realistic workload. From the ESP content, a keystroke event can be identified if they trigger system calls. As a result, we can accurately determine inter-keystroke timings and launch a timing attack to infer the characters the victim entered.

We developed techniques for automatically analyzing an application's binary executable to extract the ESP pattern that fingerprints a keystroke event. The occurrences of such a pattern are identified from an ESP trace the shadow program records from the application's runtime to calculate timings. These timings are further analyzed using a Hidden Markov Model and other public information related to the victim on a multi-user system. Our experimental study demonstrates that our attack greatly facilitates password cracking and also works very well on recognizing English words.

1 Introduction

Multi-user operating systems and application software have been in use for decades and are still pervasive today. Those systems allow concurrent access by multiple users so as to facilitate effective sharing of computing resources. Such an approach, however, is fraught with security risks: without proper protection in place, one's sensitive information can be exposed to unintended parties on the same system. This threat is often dealt with by an access control mechanism that confines each user's activities to her compartment. As an example, programs running in a user's account are typically not allowed to touch the data in another account without the permission of the owner of that account. The problem is that different users do need to interact with each other, and they usually expect this to happen in a convenient way. As a result, most multi-user systems tend to trade security and privacy for functionality, letting certain information go across the boundaries between the compartments. For example, the process status command `ps` displays the information of currently-running processes; while this is necessary for the purpose of system administration and collaborative resource sharing, the command also enables one to peek into others' activities such as the programs they run.

In this paper, we show that such seemingly minor information leaks can have more serious consequences than the system designer thought. We present a new attack in which a malicious user can eavesdrop on others' keystrokes using nothing but her non-privileged account. Our attack takes advantage of the information disclosed by procfs [19], the process file system supported by most Unix-like operating systems such as Linux, BSD, Solaris and IBM AIX. Procfs contains a hierarchy of virtual files that describe the current kernel state, including statistical information about the memory of processes and some of their register values. These files are used by the programs like `ps` and `top` to collect system information and can also help software debugging. By default, many of the files are readable for all users of a system, which naturally gives rise to the concern whether their contents could disclose sensitive user information. This concern has been confirmed by our study.

The attack we describe in this paper leverages the

procfs information of a process to infer the keystroke inputs it receives. Such information includes the contents of the extended stack pointer (ESP) and extended instruction pointer (EIP) of the process, which are present in the file `/proc/pid/stat` on a Linux system, where `pid` is the ID of the process. In response to keystrokes, an application could make system calls to act on these inputs, which is characterized by a sequence of ESP/EIP values. Such a sequence can be identified through analyzing the binary executables of the application and used as a pattern to fingerprint the program behavior related to keystrokes. To detect the keystroke event at runtime, we can match the pattern to the ESP/EIP values acquired through continuously reading from the `stat` file of the application's process. As we found in our research, this is completely realistic on a multi-core system, where the program logging those register values can run side by side with its target process. As such, we can figure out when a user strokes a key and use inter-keystroke timings to infer the key sequences [26]. This attack can be automated using the techniques for automatic program analysis [20, 23].

Compared with existing side-channel attacks on keystroke inputs [26, 3], our approach significantly lowers the bar for launching a successful attack on a multi-user system. Specifically, attacks using keyboard acoustic emanations [3, 33, 2] require physically implanting a recording device to record the sound when a user's typing, whereas our attack just needs a normal user account for running a non-privileged program. The timing attack on SSH proposed in the prior work [26] estimates inter-keystroke timings from the packets transmitting passwords. However, these packets cannot be deterministically identified from an encrypted connection [13]. In contrast, our attack detects keystrokes from an application's execution, which is much more reliable, and also works when the victim uses the system locally. Actually, we can do more with an application's semantic information recovered from its executable and procfs. For example, once we observe that the same user runs the command `su` multiple times through SSH, we can assume that the key sequences she entered in these interactions actually belong to the same password, and thus accumulate their timing sequences to infer her password, which is more effective than using only a single sequence as the prior work [26] does. As another example, we can even tell when a user is typing her username and when she inputs her password if these two events have different ESP/EIP patterns in an application.

This paper makes the following contributions:

- *Novel techniques for determining inter-keystroke timings*. We propose a suite of new techniques that accurately detects keystrokes and determines inter-keystroke timings on Linux. Our approach includes an automatic program analyzer that extracts from the binary executable of an application the instructions related to keystroke events, which are used to build a pattern that fingerprints the events. During the execution of the application, we use a shadow program to log a trace of its ESP/EIP values from procfs. The trace is searched for the occurrences of the pattern to identify inter-keystroke timing. Our attack does not need to change the application under surveillance, and works even in the presence of address space layout randomization [29] and realistic workloads. Our research also demonstrates that though other UNIX-like systems (e.g., FreeBSD and OpenSolaris) do not publish these register values, they are subject to similar attacks that utilize other information disclosed by their procfs.
- *Keystroke analysis*. We augmented the existing keystroke analysis technique [26] with semantic information: once multiple timing sequences are found to be associated with the same sequence of keys, our approach can combine them together to infer these keys, which turns out to be very effective. We also took advantage of the information regarding the victim's writing style to learn the English words she types.
- *Implementation and evaluations*. We implemented an automatic attack tool and evaluated it using real applications, including `vim`, `SSH` and `Gedit`. Our experimental study demonstrates that our attack is realistic: inter-keystroke timings can be reliably collected even when the system is under a realistic workload. We also discuss how to defend against this attack.

The attack we propose aims at keystroke eavesdropping. However, the privacy implication of disclosing the ESP/EIP information of other users' process can be much more significant. With our techniques, such information can be conveniently converted to a system-call sequence that describes the behavior of the process, and sometimes, the data it works on and the activities of its users. As a result, sensitive information within the process can be inferred under some circumstances: for example, it is possible to monitor a key-generation program to deduce the secret key it creates for another user, because the key is computed based on random activities within a system, such as mouse moves, keystrokes and networking events, which can be discovered using our techniques.

The information-leak vulnerability exploited by our attack is pervasive in Linux: we checked 8 popular distributions (Red Hat Enterprise, Debian, Ubuntu, Gentoo, Slackware, openSUSE, Mandriva and Knoppix) that represent the mainstream of Linux market [9] and found that all of them publish ESP and EIP. Some other Unix-

like systems, particularly FreeBSD, have different implementations of procfs that do not disclose the contents of those registers to unauthorized party. However, given unrestricted access to procfs, similar attacks that use other information can still happen: for example, we found that `/proc/pid/status` on FreeBSD reveals the accumulated kernel time consumed by the system calls within a process; such data, though less informative than ESP/EIP, could still be utilized to detect keystrokes in some applications, as discussed in Section 6.2. Fundamentally, we believe that the privacy risks of procfs need to be carefully evaluated on multi-core systems, as these systems enable one process to gather information from other processes in real time.

The rest of the paper is organized as follows. Section 2 presents an overview of our attack. Section 3 elaborates our techniques for detecting inter-keystroke timings. Section 4 describes a keystroke analysis using the timings. Section 5 reports our experimental study. Section 6 discusses the limitations of our attack, similar attacks on other UNIX-like systems and potential defense. Section 7 surveys the related prior research, and Section 8 concludes the paper.

2 Overview

This section describes our attack at a high level.

Attack phases. Our attack has two phases: first, the timing information between keystrokes is collected, and then such information is analyzed to infer the related key sequences. These phases and their individual components are illustrated in Figure 1. In the first phase, our approach analyzes the binary executable of an application to extract the ESP/EIP pattern that characterizes its response to a keystroke event, and samples the `stat` file of the application at its runtime to log a trace of those register values. Inter-keystroke timings are determined by matching the pattern to the trace. In the second phase, these timings are fed into an analysis mechanism that uses the Hidden Markov Model (HMM) to infer the characters being typed.

An example. We use the code fragment in Figure 2 as an example to explain the design of the techniques behind our attack. The code fragment is part of an editor program[1] for processing a keystroke input. Upon receiving a key, the program first checks its value: if it is '`MOV_CURSOR`', a set of API calls are triggered to move the cursor; otherwise, the program makes calls to insert the input letter to the text buffer being edited and display its content. These two program behaviors produce two different system call sequences, as illustrated in the figure. This example is written in `C` for illustration purpose. Our techniques actually work on binary executables.

```
1 ...
2 if (input_ready()) {
3   c = vgetc();
4   switch (c) {
5     ...
6     MOV_CURSOR: {
7       ...
8       cursor_pos_info();
9       update_cursor();
10      ...
11       ...
12    };
13    default: { //insert a char
14      ...
15      alloc_buf();
16      insert_char();
17      update_undo();
18      flush_buffers;
19      ...
20    }
21  }
22  ...
23 }
```

System Call Sequence for MOV_CURSOR:	System Call Sequence for insert a char:
read	read
select	select
select	select
select	select
select	select
select	select
select	select
select	select
select	select
select	write
write	select
select	_llseek
selec	write
	select
	fsync
	select
	select

Figure 2: An Example.

To prepare for an attack, our approach first performs a dynamic analysis on the program's executable to extract its ESP/EIP pattern that characterizes the program's response to a keystroke input. Examples of such a response includes allocating a buffer to hold the input (`alloc_buf()`) and inserting it to the text (`insert_char()`). In our research, we found that such a pattern needs to be built upon system calls because sampling of a process's `stat` file can hardly achieve the frequency necessary for catching the ESP/EIP pairs unrelated to system calls (Section 3.1). When a system call happens, the EIP of the process always points to virtual Dynamic Shared Object (vDSO)[2] [22], a call entry point set by the kernel, whereas its ESP value reflects the dynamics of the process's call stack. Therefore, our approach uses the ESP sequence of system calls as the pattern for keystroke recognition. Such a pattern is automatically identified from the executable through a differential analysis or an instruction-level program analysis (Section 3.1).

When the program is running on behalf of the victim, our approach samples its `stat` file to get its ESP/EIP values, from which we remove those unrelated to system calls according to their EIPs. The rest constitutes an ESP trace of the program's system calls. This trace is searched for the ESP patterns of keystrokes. Note that the trace may only contain part of the patterns: in the example, inserting a character triggers 17 system calls, whereas only 5 - 6 of them appear in the trace. Our approach uses a threshold to determine a match (Section 3.3). Inter-keystroke timings are measured between two successive occurrences of a same pattern.

The timings are analyzed using an n-Viterbi algorithm [26] to infer the characters being typed: our approach first constructs an HMM based upon a set of train-

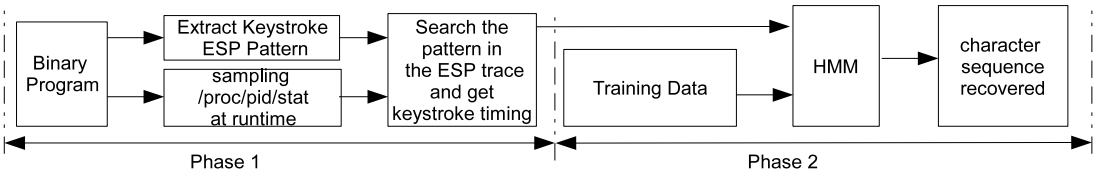

Figure 1: Attack phases

ing data that reflect the timing distributions of different key pairs the victim types, and then runs the algorithm to compute n most likely key sequences with regards to the timing sequence acquired from the ESP trace. We extend the algorithm to take advantage of multiple traces of the same key sequence, which turns out to be particularly effective for password cracking. We also show that the techniques are also effective in inferring English words a user types.

Assumptions. We made the following assumptions in our research:

- *Capability to execute programs.* To launch the attack, the attacker should own or control an account that allows her to execute her programs. This is not a strong assumption, as most users of UNIX-like systems do have such a privilege. The attacker here could be a malicious insider or an intruder who cracks a legitimate user's account.
- *Multi-core systems.* To detect a keystroke, our shadow process needs to access the ESP of the target process before it accomplishes key-related system calls. However, due to process scheduling, this is not very likely to happen on a single-core system. On one hand, these system calls are typically done within a single time slice. On the other hand, the shadow process often lacks sufficient privileges to preempt the target process when it is working on keystroke inputs, as the latter is usually granted with a high privilege during its interactions with the user. As a result, our process can become completely oblivious to the keystroke events in the target process. This problem is effectively avoided on a multi-core system, which allows us to reliably detect keystroke events in the presence of realistic workloads[3], as observed in our experiment (Section 5). Given the pervasiveness of multi-core systems nowadays, we believe that the assumption is reasonable.
- *Access to the victim's information.* Our attack requires a read access to the victim's procfs files. This assumption is realistic for Linux, on which most part of procfs are readable for every user by default. Though one can change her files' permissions, this can hardly eliminate the problem: all the procfs files are dynamically created by the kernel when a new process is forked and their default permissions are also set by the kernel; as a result, one needs to revise these permissions as soon as she triggers new process, which is unreliable and also affects the use of the tools such as `top`. The fundamental solution is to patch the kernel, which has not been done yet. In addition, we assume that the attacker can obtain some of the text the victim types as training data. This is possible on a multi-user system. For example, some commands typed by a user, such as "`su`" and "`ls`", causes new processes to be forked and therefore can be observed by other users of the system, which allows the observer to bind the timing sequence of the typing to the content of the text the user entered. As another example, a malicious insider can use the information shared with the victim, such as the emails they exchanged, to acquire the latter's text and the corresponding timings.

3 Inter-keystroke Timing Identification

In this section, we elaborate our techniques for obtaining inter-keystroke timings from a process.

3.1 Pattern Extraction

The success of our attack hinges on accurate identification of keystroke events from the victim's process. We fingerprint such an event with an ESP pattern of the system calls related to a keystroke. The focus on system calls here comes from the constraints on the information obtainable from a process: on one hand, a significant portion of the process's execution time can be spent on system calls, particularly when I/O operations are involved; on the other hand, our approach collects the process's information through system calls and therefore cannot achieve a very high sampling rate. As a result, the shadow program that logs ESP/EIP traces is much more likely to pick up system calls than other instructions. In our research, we found that more than 90% of the ESP/EIP values collected from a process actually belong to system calls. Note that a process's EIP when it is making a system call always points to vDSO. It is used in our research to locate the corresponding ESP whose content is much more dynamic and thus more useful for fingerprinting a keystroke event.

Our approach extracts the ESP pattern through an automatic analysis of binary executables. This analysis is conducted offline and in an environment over which the attacker has full control. Following we present two analysis techniques, one for the programs that execute in a deterministic manner and the other for those whose executions are affected by some random factors.

Differential analysis. Many text-based applications such as `vim` are deterministic in the sense that two independent runs of these applications under the same keystroke inputs yield identical system call traces and ESP sequences. The ESP patterns of these applications can be easily identified through a differential analysis that compares the system call traces involving keystroke events with those not. Specifically, our program analyzer uses `strace` [27] to intercept the system calls of an application and record their ESP values when it is running. An ESP sequence is recorded before a keystroke is typed, and another sequence is generated after the keystroke occurs[4]. The ESP pattern for a keystroke event is extracted from the second sequence after removing all the system calls that happen prior to the keystroke, as indicated by the first sequence. To ensure that the pattern does not contain any randomness, we can compare the ESP trace of typing the same character twice with the one involving only a single keystroke to check whether the ESPs associated with the second keystroke are identical to those of the first one. The same technique is also applied to test different keys that may have discrepant patterns. In the example described in Figure 2, the ESP sequence of `vim` before Line 2 is dropped from the traces involving keystrokes and as a result, the system calls triggered by the instructions from Line 7 to 11 are picked out as the fingerprint for '`MOV_CURSOR`' and those between Line 14 and 19 identified as the pattern for inserting a letter.

The ESP pattern identified above will go through a false positive check to evaluate its accuracy for keystroke detection. In other words, we want to know whether the pattern or a significant portion of it can also be observed when the user is not typing. This is achieved in our research through searching for the pattern in an application's ESP trace unrelated to keystroke inputs. Specifically, our analyzer logs the execution time between the first and the last system calls on the pattern, and uses this time interval to define a duration window on the trace, which we call *trace window*. The trace window is slid on the trace to determine a segment against which the pattern is compared. For this purpose, every ESP value on the trace is labeled with the time when its corresponding system call is invoked. The trace window is first located prior to the first ESP value on the trace. Then, it is slid rightwards: each slide either moves an ESP into the window or moves one outside the window. After a slide, our analyzer attempts to find the longest common sequence between the trace segment within the window and the pattern. This is the well-known LCS problem [4], which can be efficiently solved through dynamic programming [15]. The size of such a sequence, which we call an *FP level*, is recorded. As such, our approach keeps on sliding the trace window to measure FP levels until all the ESP values on the trace have left the window.

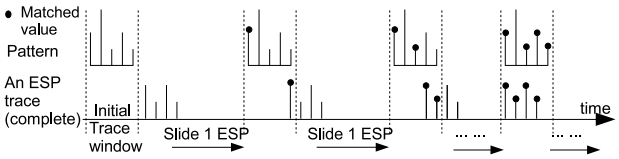

Figure 3: A false positive check. Spikes in the figure represent ESP values.

Figure 3 presents an example that shows how the algorithm works. In the initial state, the trace window is located before the first ESP value. Then the trace window starts to slide right to include the first ESP value, which gives a FP level of one. After the window slides again to include one more ESP value, our algorithm returns a common sequence with two members. This process continues, and finally, the window is moved to embrace all four trace members and we observe an FP level of four. This algorithm identifies the portion of the pattern that can show up in absence of keystrokes. The size of the portion, as indicated by the FP level, is used to determine a threshold for recognizing keystrokes from an incomplete ESP trace sampled from a process, which is elaborated in Section 3.3.

Instruction-level analysis. Applications with graphic user interfaces (GUI) can work in a non-deterministic manner: these applications are event-driven and can change their system-call behaviors in response to the events from operating systems (OS), which can be unpredictable. For example, `Gedit` uses a timer to determine when to flash its cursor; the timer, however, can be delayed when the process is switched out of the CPU, which causes system call sequences to vary in different runs of the application. To extract a pattern from these applications, we adopted an instruction-level analysis as described below.

Under Linux, many X-Window based applications are developed using the GIMP Toolkit (aka. GTK+) [28]. GTK+ uses a standard procedure to handle the keystroke event: a program uses a function such as `gtk_main_do_event(event)` to process `event`; when a key is pressed[5], this function is invoked to trigger a call-back function of the keystroke event. In our research, we implemented a Pin [20] based analysis tool that automatically analyzes a binary executable at the instruction level to identify such a function. After a key has been typed, our analyzer detects the keystroke

event from the function's parameter and from that point on, records all the system calls and their ESPs until the executable is found to receive or dispatch a new event, as indicated by the calls to the functions like `g_main_context_acquire()`. All these system calls are thought to be part of the call-back function and therefore related to the keystroke event[6]. The pattern for keystroke recognition is built upon these calls. We also check false positives of the pattern, as described before.

3.2 Trace Logging

Our attack eavesdrops on the victim's keystrokes through shadowing the process that receives her keystroke inputs. Our shadow process stealthily monitors the target process's keystroke events by keeping track of its ESP/EIP values disclosed by its `stat` file. Since the attack happens in the userland, the attacker has to use system calls to open and read the file. Moreover, a more efficient approach, memory mapping through `mmap()`, does not work on the virtual file that exists only in memory. These issues prevent the shadow process from achieving a high sampling rate. For example, a program we implemented for evaluating our approach updated ESP/EIP values every 5 to 10 microseconds. As a result, we could end up with an incomplete ESP/EIP trace of the target process. This, however, is sufficient for determining inter-keystroke timings, as we found in our research (Section 3.3).

Trace logging with full steam can cost a lot of CPU time. If the activity drags on, suspicions can be roused and alarms can be triggered. To avoid being detected, our attack takes advantage of the semantic information recovered from procfs and the target application to concentrate the efforts of data collection on the time interval when the victim is typing the information of interest to the attacker. For example, the shadow process starts monitoring the victim's `SSH` process at a low rate, say once per 100 milliseconds; once the process is observed to fork a `su` process, our shadow process immediately increases its sampling rate to acquire the timings for the password key sequence. Another approach is using an existing technique [32] to hide CPU usage: UNIX-like systems keep track of a process's use of CPU according to the number of ticks it consumes at the end of each tick; the trick proposed in [32] lets the attack process sleep just before the end of each tick it uses and as a result, OS will schedule a victim process to run and bill the whole tick to that victim process instead of the attack process. We implemented this technique and found that it was very effective (Section 5).

3.3 Timing Detection

We determine inter-keystroke timings from the time intervals between the occurrences of a pattern on an ESP trace sampled from an application's system calls. Two issues here, however, complicate the task. First, some Linux versions may run the mechanisms for address space layout randomization (ASLR) [29] that can cause the ESP values on the pattern to differ from those on the trace. Second, the trace can be incomplete, containing only part of the system calls on the pattern, which makes recognition of the pattern nontrivial. Following we show how these issues were handled in our research.

ASLR performed by the tools such as `Pax` [30] involves randomly arranging the locations of an executable's memory objects such as stack, executable image, library images and heap. It is aimed at thwarting the attacks like control-flow hijacking that heavily rely on an accurate prediction of target memory addresses. Though the defense works on the attacks launched remotely, it is much less effective on our attack, which is commenced locally. Specifically, the address for the bottom of a process's stack can be found in its `stat` and `/proc/PID/maps`[7]. This allows us to "normalize" the ESP values on both the trace and the pattern with the differences between the tops of the stack, as pointed by the ESPs, and their individual bottoms. Neither does ASLR prevent us from correlating an ESP/EIP pair on a trace to a system call, though the knowledge about the vDSO address may not be publically available on some Linux versions: we can filter out the pairs unrelated to system calls according to the observation that the vast majority of the members on the trace actually belong to system calls and therefore have the same EIP values.

To recognize an ESP pattern from an incomplete ESP trace of system calls, we use a threshold τ: a segment of the trace, as determined by the trace window, is deemed matching the pattern if it contains at least τ ESP values of system calls and the sequence of these values also appear on the pattern. The threshold here can be determined using the results of the false positive test described in Section 3.1. Let h be the highest FP level found in the test, and s be the number of the system calls that our shadow process can find from a process when a keystroke occurs. We let $\tau = h + 1$ if $s > h$. Intuitively, this means that a trace segment is considered matching the pattern if it does not contain any ESP sequences not on the pattern and no segments unrelated to keystrokes can match as many ESP values on the pattern as that segment does[8]. If $s \leq h$, we have to set $\tau = s$ because we cannot get more than s ESP samples for every keystroke when monitoring a process. Several measures can be taken to mitigate the false positives that threshold could bring in. One approach is to leverage the observation that people typ-

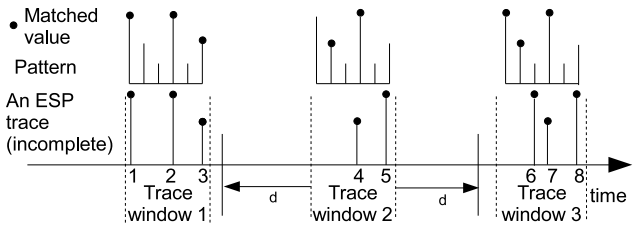

Figure 4: Using time frame d to remove possible false positive matches

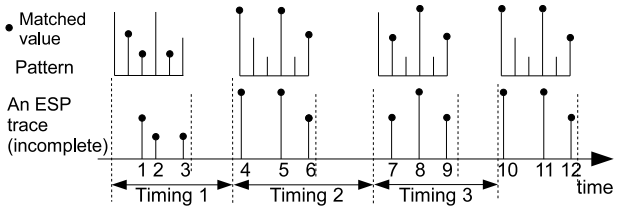

Figure 5: Pattern matching on an ESP trace and the timing interval

ically type more than one key within a short period of time. Therefore, we can require that a segment matching a pattern according to τ be preceded or followed by another pattern-matching segment within a predetermined time frame d, before both of them can be deemed to be indicative of keystroke events. Figure 4 presents an example in which the segment within the Window 2 is not treated as a match to the pattern because there is no other matches happening within the time frame d either before or after the window. In another approach, we use the execution time of a process to estimate the time point when it starts receiving keystrokes, which helps avoid searching the trace unlinked to keystrokes.

After normalizing ESP values and determining the threshold τ, our approach starts searching the trace sampled from the victim's process for the occurrences of the pattern. The searching algorithm we adopted slides the trace window in the same way as the false positive check does (Section 3.1). For each slide, an LCS problem is solved to find the longest common sequence between the trace segment in the window and the pattern. If the length of the sequence is no less than τ and every member on the segment is also on the sequence, the segment is labeled as a match. Once a match is found, we slide the window rightwards to pass all trace members within a short time interval that describes the minimal delay between two consecutive keystrokes, and then start the next round of searching. This process continues until all trace members pass the window. Then, our approach determines timings from the segments labeled as matches: the time interval between two such segments is identified as an inter-keystroke timing if there is no other labeled segments in-between and the duration of the interval is below a predetermined threshold that serves to rule out the long latencies caused by intermittent typing. An example for illustrating the algorithm is presented in Figure 5, in which the trace window locates four matches with $\tau = 3$, and the durations between these matches are picked out as inter-keystroke timings.

4 Keystroke Analysis

In this section, we describe how to use inter-keystroke timings to infer the victim's key sequence. Our approach is built upon the technique used in the existing timing attack [26]. However, we demonstrate that the technique can become much more effective with the information available on a multi-user system.

4.1 HMM-based Inference of Key Sequences

A Hidden Markov Model [24] describes a finite stochastic process whose individual states cannot be directly observed. Instead, the outputs of these states are visible and therefore can be used to infer the existence of these states. An HMM, like a regular Markov model, assumes that the next states a system can move into only depend on the current state. In addition, it has a property that the outputs of a state are completely determined by that state. These two properties allow a hidden sequence to be easily computed and therefore make the model a pervasive tool for the purposes such as speech recognition and text modeling.

Prior research [26] models the problem of key inference using an HMM. Specifically, let $K_0, \ldots, K_T$ be the key sequence typed by the victim, and $q_t \in Q$ $(1 \leq t \leq T)$ be a sequence of states representing the key pair (K_{t-1}, K_t), where Q is the set of all possible states. In each state q_t, an inter-keystroke latency y_t with a Gaussian-like distribution can be observed. Our objective is to find out the hidden states $(q_1, \ldots, q_T)$ from the timings $(y_1, \ldots, y_T)$. This modeling is simple and was shown to work well in practice [26], and is further confirmed by our research, though it has oversimplified the relations between the characters being typed: particularly, the chance for a letter to appear at a certain position in an English word may actually relate to all other letters before it, which invalidates the HMM assumption that a transition from q_t to q_{t+1} depends only on q_t.

The HMM for key inference can be solved using the Viterbi algorithm [24], a dynamic programming algorithm that computes the most likely state sequence $(q_1, \ldots, q_T)$ from the observed timing sequence $(y_1, \ldots, y_T)$. Let $V(q_t)$ be the probability of the sequence that most likely ends in q_t at time t. The algorithm computes $V(q_t)$ through two steps. In the first step, we assign a set of initial probabilities $V(q_1) =$

$Pr[q_1|y_1]$. The second step inductively computes $V(q_t)$ for every $1 < t \leq T$ and every $q_t \in Q$ as $V(q_t) = \max_{q_{t-1}} Pr[y_t|q_t]Pr[q_t|q_{t-1}]V(q_{t-1})$, where $Pr[y_t|q_t]$ can be estimated from a set of training data (the third assumption in Section 2) and $Pr[q_t|q_{t-1}]$, the transition probability, comes from a uniform distribution over the states reachable from q_{t-1}. This step also keeps track of all the prior states on the sequence with the probability $V(q_t)$. The most likely sequence is identified from the state q_T that maximizes $V(q_T)$. A direct application of this approach, however, does not work well in practice, because even the most likely sequence usually has a very small probability to match the real keystroke inputs. This problem is mitigated in the prior work [26] that extends the algorithm to the n-Viterbi algorithm so as to return the top n most likely sequences given a timing sequence. The difference here is that the n-Viterbi algorithm changes the inductive step (the second step) to identify the sequences with the n largest probabilities. The details of the algorithm can be found in [26].

4.2 Password Cracking

The effectiveness of the n-Viterbi algorithm can be significantly improved with the information available on a multi-user system. Particularly, the name of a process and its owner can be directly found from procfs or indirectly from running commands such as `ps` or `top`. Once the same user is observed to run the same application multiple times and if such interactions happen within a no-so-long period of time and all involve typing passwords, a reasonable assumption we can make is that all these passwords are actually the same. Therefore, we can combine together the timing sequences recorded from individual interactions to infer a key sequence. Following we describe two ways to do that.

Our first approach is simply averaging all the timings for every key pair to create a new sequence and run the n-Viterbi algorithm over it. Formally, given m timing sequences $(y_1^1, \ldots, y_T^1), \ldots, (y_1^m, \ldots, y_T^m)$, we can compute a new sequence $(y_1, \ldots, y_T)$, where $y_t = \frac{1}{m}\sum_{1\leq i\leq m} y_t^i$ and $1 \leq t \leq T$. The rationale here is that the distribution of the timing y_t^i of a key pair q_t is a Gaussian-like unimodal distribution and therefore the probability $Pr[y_t|q_t]$ in the inductive step of the algorithm is maximized when y_t becomes the mean of the distribution, which is approximated by averaging all y_t^i. This approach works particularly well when the means of two key pairs are not extremely close.

The other approach, which we call the m-n-Viterbi algorithm, utilizes multiple observations to perform the inductive step of the original algorithm. Specifically, our approach replaces $Pr[y_t|q_t]$ in that step with $Pr[y_t^1, \ldots, y_t^m|q_t] = Pr[y_t^1|q_t] \ldots Pr[y_t^m|q_t]$ given these observations $(y_t^1, \ldots, y_t^m)$ are independent from each other. This treatment works even in the presence of the key pairs with very close timing distributions. However, it needs a large number of timing sequences to get a good outcome.

Our research shows that both approaches can significantly shrink the space for searching a password. Actually, in our experiment (Section 5.2), we found that using 50 timing sequences, our techniques sped up the password searching by factors ranging from 250 to 2000.

4.3 English Text

Recovery of English text from a timing sequence is no less challenging than password cracking. A password can be figured out through testing many candidates against the target application or a hashed password list. However, the same trick cannot be played on English words because no application and password list can tell you whether you made a right guess. All that we can do is to check all the combinations of the possible words to see whether a meaningful sentence comes out, which becomes a daunting task if the list of such words is long. Moreover, it can be more difficult to find multiple timing sequences associated with the same text, and therefore the aforementioned approaches become less applicable. On the bright side, English words are much less random than passwords: the letters they include and the combinations of those letters have distributions with low entropies. Such a property can be leveraged to adjust the transition probabilities of an HMM to improve the outcomes of key sequence inference. Here we elaborate such techniques used in our research.

A prominent property of English text is use of the SPACE character to separate words. People tend to type the letters in a word faster than SPACE, a signal for a transition between words. This gives the character an identifiable timing feature: typically the key pair involving SPACE incurs longer inter-keystroke latency than other pairs, as illustrated in Figure 6. In our research, we detected SPACE by checking if the timing interval is larger than a predetermined threshold. This threshold can be determined from the training data collected from the victim's typing. Knowledge about the SPACE key helps us to divide a long timing sequence into a collection of small sequences, with each of them representing a word, and then learn these words one by one.

Another important property of English text is its distinct distribution of letters. It is well known that some letters such as 'e' occur more frequently than others, and some bigrams like 'th' and trigrams like 'ion' are also pervasive in a meaningful text. This fact has been leveraged by frequency analysis to crack classic ciphers [1]. The same game can also be played to make key se-

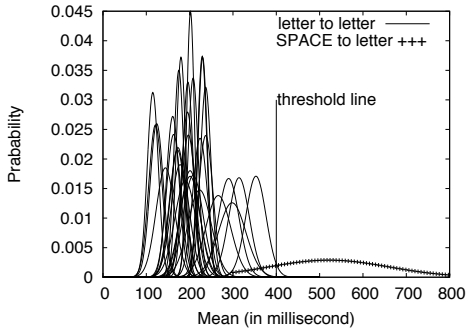

Figure 6: Timing Distribution of SPACE-letter pair, letter-letter pair and threshold

quence inference more effective: we can adjust the transition probabilities of an HMM to ensure that the transition between certain states such as ('i','o') to ('o', 'n') is more likely to happen than others. These probabilities can be conveniently obtained from various public sources [18, 10] that provide the statistics of common English text. Such statistics can be further tuned to the victim's writing style according to public writing samples such as her web pages and publications. Moreover, it comes with no surprise that users on the same system are often related: for example, they could all belong to one organization. This allows the attacker to get familiar with the victim's writing from the information they exchanged, for example, the emails between them. In addition, since the timing sequence corresponding to such information can also be identified using our technique, the attacker can actually use the information as the training data for estimating the timing distributions of different key pairs the victim typed.

5 Evaluation

In this section, we describe an experimental study of the attack techniques we propose. Our objective is to understand whether these techniques present a realistic threat. To this end, we evaluated them using 3 common Linux applications: `vim`, `SSH` and `Gedit`. In our experiments, we first ran our approach to automatically extract timing sequences when a user was typing, evaluated the accuracy of these timings and the effectiveness of the attack under different workloads. Then, we analyzed them using our techniques to study how much keystroke information could be deduced. Our experiments were mainly carried out on a computer with a 2.40GHz Core 2 Duo processor and 3GB memory, on which we conducted our study under three Linux versions: RedHat Enterprise Linux 4.0, Debian 4.0 and Ubuntu 8.04. We found that our techniques worked effectively even in the presence of realistic workloads on the server. This suggests that the information leaks caused by procfs can be a real security problem.

Table 1: Normalized ESP pattern values (include system calls)

vim		ssh		gedit	
SysCall	ESP	SysCall	ESP	SysCall	ESP
read	1628	rt_sigprocmask	4932	gettimeofday	3624
select	1604	rt_sigprocmask	4932		
select	1876	read	20908		
select	2244	select	4548		
select	1540	rt_sigprocmask	4932		
select	1908	rt_sigprocmask	4932		
select	1556	write	37436		
select	1924	ioctl	37500		
select	1604	select	4548		
write	1548	rt_sigprocmask	4932		
select	1972	rt_sigprocmask	4932		
_llseek	1876	read	37436		
write	1836	select	4548		
select	2180	rt_sigprocmask	4932		
fsync	1752	rt_sigprocmask	4932		
select	2148	write	4620		
select	1972	select	4548		

5.1 Inter-keystroke Timings

As the first step of our evaluation, we applied our technique to identify the timings from `vim`, `SSH` and `Gedit` on a multi-core system.

`vim`. `vim` is an extremely common text editor, which is supported by almost all Linux versions. It fits well with the notion of deterministic programs as discussed in Section 3.1, because independent runs of the application with the same inputs always produce the same system call sequence and related ESP sequence. This property enabled us to identify its ESP pattern for a keystroke event using the differential analysis. The pattern we discovered for inserting a letter includes 17 calls. These calls and their normalized ESP values are presented in Table 1. We further ran the application from a user account to enter words, and in the meantime, launched a shadow process from another account to collect the ESP trace of the application. From the trace, our approach automatically identified all the keystrokes we typed. Table 2 shows a trace segment corresponding to two keystrokes, which involves 5 system calls for each keystroke.

In order to evaluate the accuracy of the timing sequence our shadow process found, an instrumented version of `vim` was used in our experiment, which recorded the time when it received a key from `vgetc()`. Such information was used to compute a real timing sequence. We compared these two sequences and found that the de-

Table 2: Examples of ESP traces (values that appear in the pattern are in bold font).

vim	ssh	gedit
1604	**4548**	520
2244	**4932**	2988
1908	**20908**	3052
1924	**4548**	696
1972	**37500**	**3624**
1604	**4548**	3068
2244	**37436**	2988
1908	**4932**	696
1924	**4620**	520
1972	**4548**	2988

viations between corresponding timings were at most 1 millisecond, below 3% of the average standard deviation of the timings of different key pairs, as illustrated in Table 3. This demonstrates that the timings extracted from the process were accurate.

SSH. The Secure Shell (SSH) has long been known to have a weakness in its interactive mode, where every keystroke is transmitted through a separate packet and immediately after the key is pressed. This weakness can be exploited to determine inter-keystroke timings for inferring the sensitive information a user types, such as the password for su. Prior work [26] proposes an attack that eavesdrops on an SSH channel to identify such timings. A problem of the attack, as pointed out by SSH Communications Security, is that determination of where a password starts in an encrypted connection can be hard [25]. This problem, however, does not present a hurdle to our attack, because we can easily find out from procfs when su is spawned from an SSH process, and start collecting information from SSH from then on. This is exactly what we did in our experiment.

Using the differential analysis, our approach automatically discovered an ESP pattern from SSH when a key was typed for entering a password for su. We further ran a shadow process to monitor another user's SSH process: as soon as it forked an su process, our shadow process started collecting ESP values from the SSH process's stat file. The trace collected thereby was compared with the pattern to pinpoint keystroke events and gather the timings between them. The pattern that we found in our experiment included 17 system calls, of which 7 to 10 appeared in every occurrence of the pattern on the trace. The detailed experimental results are in Table 1 and Table 2.

Verification of the correctness of those timings turned out to be more difficult than we expected. su does not read password characters one by one from the input. Instead, it takes all of them after a RETURN key has been stroked. Therefore, instrumentation of its source code will not give us the real timing sequence. We solved this problem by replacing su with another program that recorded the time when it received a key from SSH, and used such information to generate a timing sequence. This sequence was found to be very close to the one we got from the trace collected by our shadow process, as described in Table 3. We further employed the timings obtained from su to infer the passwords being typed, which we found to be very effective (Section 5.2).

Table 3: Examples of the timings measured from ESP traces (Measured) and the real timings (Real) in milliseconds.

Timings	vim		ssh		Gedit	
	measured	real	measured	real	measured	real
1	80	81	135	135	301	303
2	139	139	124	123	285	285
3	88	88	103	103	259	259
4	101	101	110	109	236	236
5	334	335	134	134	181	182
6	86	87	111	110	265	265
7	124	124	132	132	174	174

Gedit. Gedit is a text editor designed for the X Window system. Like many other applications based upon the GTK+, it is non-deterministic in the sense that two independent runs of the application under the same inputs often produce different system call sequences. In our experiment, we performed an instruction-level analysis of its binary executables using the Pin-based tool we developed. This analysis revealed the call-back function of the key-press event, from which we extracted the system call sequence and related ESP sequence. An interesting observation is that Gedit actually does not immediately display a character a user types: instead, it put the character to a buffer through a GTK+ function gtk_text_buffer_insert_interactive_at_cursor(), which does not involve any system calls, and the content of the buffer is displayed when it becomes full or a timer expires. As a result, we could not count on the system calls involved in such a display process for fingerprinting keystrokes. Actually, only one system call was found to be present every time when a key was received: gettimeofday(), a call that Gedit uses to determine when to auto-save the document the user is editing. This call seems too general. However, its ESP value turned out to be specific enough for a pattern: in our false positive check, we did not find any other system calls within the application that also had the same ESP. Moreover, our shadow process always caught that ESP whenever we typed. Therefore, this ESP value was adopted as the pattern in our experiment. We further instrumented Gedit to dump the time when this call was invoked for calculating the real timing sequence. Table 1 shows that this sequence is very close to the one collected by our shadow process.

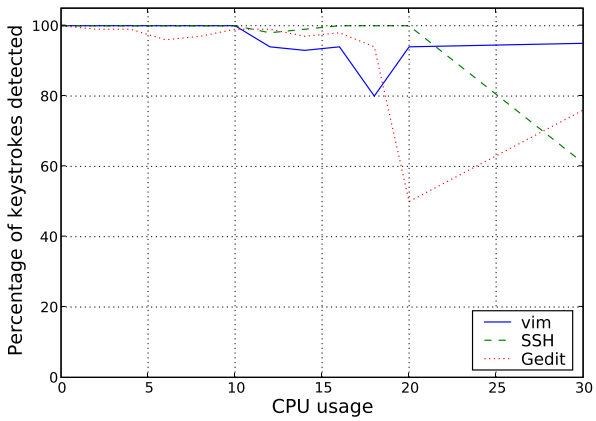

Figure 7: Percentage of keystrokes detected vs. CPU usage

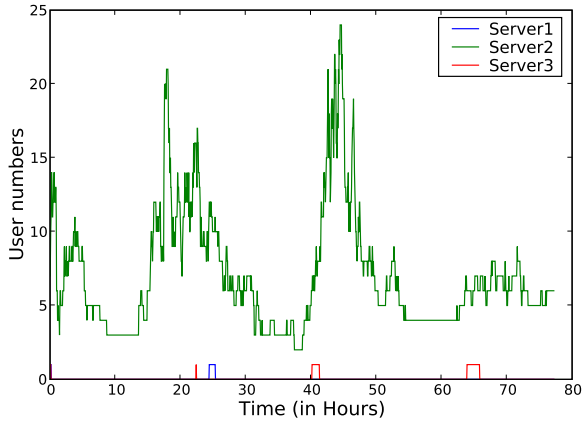

Figure 9: Variations of user numbers on the three servers during 72 hours

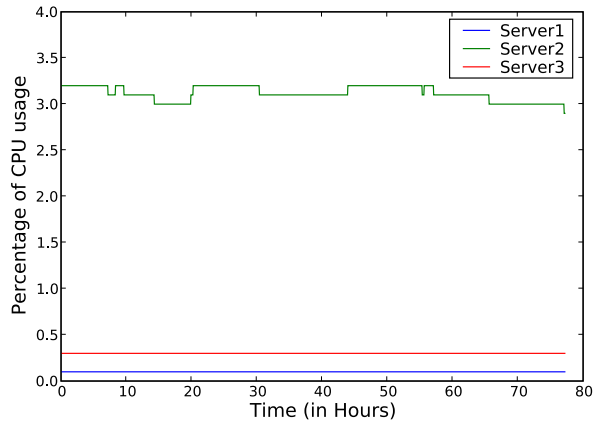

Figure 8: CPU usages of three real-world servers during 72 hours

Impacts of server workloads. A multi-user system often concurrently serves many users. These users' activities could interfere with the collection of inter-keystroke timings. This problem was studied in our research through evaluating the effectiveness of our attack under different workloads. Specifically, we ran our attacks on `vim`, `SSH` and `Gedit` under different CPU usages to measure the percentage of the keystrokes still detectable to our shadow process. The experimental results are elaborated in Figure 7. Here, we sketch our findings.

We found that the impacts of workloads varied among applications. The attacks on `vim` and `SSH` appear to be quite resilient to the interferences from other processes: our shadow process picked up 100% keystrokes for both applications when CPU usage was no more than 10% and still detected 94% from `vim` when the usage went above 20%. In contrast, the attack on `Gedit` was less robust: we started missing keystrokes when more than 2% of CPU time was consumed by other processes. This discrepancy comes from applications' ESP patterns: those involving more system calls are easier to detect.

On the other hand, the workloads on a real-world system are reasonable enough to be handled by our attack. Figure 8 and 9 reports the CPU usages and user numbers we measured from three real-world systems, including a Linux workstation in a public machine room (Server 1), a server for students' course projects (Server 3) and a web server of Indiana University that allows `SSH` connections from its users (Server 2). The number of users on these systems range from 1 to 24. Our 72-hour monitoring reveals that for 90 percent of time, the CPU usages of these servers were below 3.2%.

We also implemented the technique proposed in [32] to hide the CPU usage of our shadow process. As a result, the process appeared to consume 0% of CPU, as observed from `top`. The cost, however, was that it only reliably identified about 50% of keystrokes we entered. Nevertheless, this still helped inference of keys, particularly when the same input from a user (e.g., password) was sampled repeatedly, as discussed in Section 4.2.

5.2 Key Sequence Inference

We further studied how to use the timings to infer key sequences. Experiments were conducted in our research to evaluate our techniques using both passwords and English words. Here we report the results.

Password. To study the effectiveness of our approach on passwords, we first implemented the n-Viterbi algorithm [26] and used it to compute a baseline result, and then compared the baseline with what can be achieved by the analysis using multiple timing sequences, as described in Section 4.2. Our experiment was carefully

Table 4: The percentage of the search space the attacker has to search before the right password is found.

Method	Test Cases		
	password 1	password 2	password 3
Baseline(n-Viterbi)	7.8%	6.6%	6.8%
Timing Averaging	0.38%	0.34%	0.05%
m-n-Viterbi	0.39%	0.34%	0.05%

designed to make it comparable with that of the prior work [26]: we chose 15 keys for training and testing an HMM, which include 13 letters and 2 numbers[9]. From these keys, we identified 225 key pairs and measured 45 inter-keystroke timings for each of these pair from a user. We found that the timing for each pair indeed had Gaussian-like distributions. These distributions were used to parameterize two HMMs: one for the first 4 bytes of an 8-byte password and the other for the second half.

We randomly selected 3 passwords from the space of all possible 8-byte sequences formed by the 15 characters. For each password, we ran the n-Viterbi algorithm on 50 timing sequences. Each of these sequences caused the algorithm to produce a ranking list of candidate passwords. The position of the real password on the list describes the search space an attacker has to explore: for example, we only need to check 1012 candidates if the password is the 1012th member on the list, which reduces the search space for a 4-byte password by 50 times. To avoid the intensive computation, our implementation only output the top 4500 members from an HMM. We found that for about 75% of the sequences tested in our experiment, their corresponding passwords were among these members. In Table 4, we present the averaged percentage of the search space for finding a password.

We tested the timing averaging approach and m-n-Viterbi algorithm described in Section 4.2 with 50 timing sequences for each password, and present the results in Table 4. As the table shows, both approaches achieved significant improvements over the n-Viterbi algorithm: they shrank the search space by factors ranging from 250 to 2000. In contrast, the speed-up factor introduced by the n-Viterbi algorithm was much smaller[10].

We also found that the speed-up factors achieved by our approach, like the prior work [26], depended on the letter pairs the victim chose for her password: if the timing distribution of one pair (Figure 6) is not very close to those of other pairs, it can be more reliably determined, which contributes to a more significant reduction of searching spaces. For example, in Figure 6, a password built on the pairs whose means are around 300 milliseconds is much easier to be inferred than the one composed of the pairs around 100 milliseconds, as the latter pairs are more difficult to distinguish from others with very similar distributions. It is important to note that those distributions actually reflect an individual's typing practice, and therefore, the same password entered by one can become easier to crack than by another.

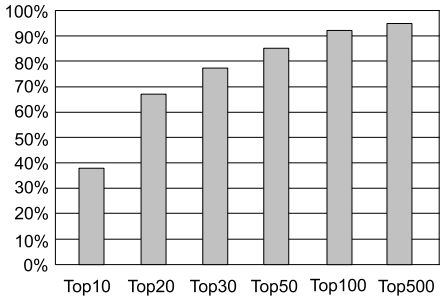

Figure 10: The success rates of the attack on English words

English words. We also studied how the timing information can help infer English words. To prepare for the experiment, a program was used to randomly generate character sequences with lengths of 3, 4 and 5 letters[11], and from them, we selected 2103 words that also appeared in a dictionary. These words were classified into three categories according to their lengths. For the words within each category, we computed a distribution using their frequencies reported by [18] . These distributions were used to determine the transition probabilities of the HMMs for individual categories, which we applied to infer the words with different lengths.

In the experiment, we randomly draw words from each category in accordance with their distribution, and typed them to collect timing sequences. The timing segments that represented individual words were identified from the sequences using the feature of the SPACE key. For each segment, we picked up an HMM according to the length of the word and solved it using the n-Viterbi algorithm, which gave us a ranking list of candidates. From the list, our approach further removed the candidates that did not pass a spelling check. We tested 14 3-letter words, 11 4-letter words and 14 5-letter words. The outcomes are described in Figure 10. From the table, we can see that the real words were highly ranked in most cases: almost 40% of them appeared in top 10 and 86% among top 50.

6 Discussion

6.1 Further Study of the Attack

Our current implementation only tracks the call-back function for the key press event. We believe that the pattern for keystroke recognition can be more specific and easier to detect by adding the ESP sequences of the system calls related to the key release event. Moreover, we evaluated our approach using three applications. It is interesting to know whether other common applications

are also subject to our attack. What we learnt from our study is that our attack no longer works when system calls are not immediately triggered by keystrokes. This could happen when the victim's process postpones the necessary actions such as access to the standard I/O until multiple keystrokes are received. For example, `su` does not read a password character by character, and instead, imports the string as a whole; as a result, it cannot be attacked when it is not used under the interactive mode of `SSH`. As another example, `GTK+` applications tend to display keys only when the buffer holding them becomes full or a timer is triggered. Further study to identify the type of applications vulnerable to our attack is left as our future research. In addition, it is conceivable that the same techniques can be applied beyond identification of inter-keystroke timing. For example, we can track the ESP dynamics caused by other events such as moving mouse to peek into a user's activities.

Our current research focuses more on extracting inter-keystroke timings from an application than on analyzing these timings. Certainly more can be done to improve our timing analysis techniques. Specifically, password cracking can be greatly facilitated with the knowledge about the types of individual password characters such as letter or number. Acquisition of such knowledge can be achieved using our enhanced versions of the n-Viterbi algorithm that accept multiple timing sequences. This "classification" attack can be more effective than the timing attack proposed in [26], as it does not need to deal with a large key-pair space. Moreover, the approach we used to infer English words is still preliminary. We did not evaluate it using long words, because solving the HMMs for these words can be time consuming. A straightforward solution is to split a long word into small segments and model each of them with an HMM, as we did for password cracking. This treatment, however, could miss the inherent relations between the segments of a word, which is important because letters in a word are often correlated. Fundamentally, the first-order HMM we adopted is limited in its capability of modeling such relations: it cannot describe the dependency relation beyond that between two key pairs. Application of other language models such as the high-order HMM [12] can certainly improve our techniques.

Actually, ESP/EIP is by no means the only information within procfs that can be used for acquiring inter-keystroke timings. Other information that can lead to a similar attack includes interrupt statistics file `/proc/interrupts`, and network status data `/proc/net`. The latter enables an attacker to track the activities of the TCP connections related to the inputs from a remote client. Moreover, the procfs of most UNIX-like systems expose the *system time* of a process, i.e., the amount of time the kernel spends serving the system calls from the process. Disclosure of such information actually enables keystroke eavesdropping, which is elaborated in Section 6.2.

6.2 Information Leaks in the Procfs of Other UNIX-like Systems

Besides Linux, most other UNIX-like systems also implement procfs. These implementations vary from case to case, and as a result, their susceptibilities to side-channel attacks also differ. Here we discuss such privacy risks on two systems, FreeBSD and OpenSolaris.

FreeBSD manages its process files more cautiously than Linux[12]: it puts all register values into the file `/proc/pid/regs` that can only be read by the owner of a process, which blocks the information used by our attack. However, we found that other information released by the procfs can lead to similar attacks. A prominent example is the system time reported by `/proc/pid/status`, a file open to every user. Figure 11 shows the correlations between the time consumed by `vim` and the keystrokes it received, as observed in our research. This demonstrates that keystroke events within the process can be identified from the change of its system time, which makes keystroke eavesdropping possible. A problem here is that we may not be able to detect special keys a user enters, for example, "`MOV_CURSOR`", which is determined from ESP/EIP information on Linux. A possible solution is using the discrepancies of system-time increments triggered by different keys being entered to fingerprint these individual keys. Further study of this technique is left to our future research.

OpenSolaris kernel makes the `/proc` directory of a process only readable to its owner, which prevents other users from entering that directory. Interestingly, some files under the directory are actually permitted to be read by others, for supporting the applications such as `ps` and `top`. Like FreeBSD, the registers of the process are kept off-limits. However, other information, including system time, is still open for grabs. Figure 11 illustrates the changes of the system time versus a series of keystrokes we entered on OpenSolaris, which demonstrates that identification of inter-keystroke timings is completely feasible on the system.

6.3 Defense

An immediate defense against our attack is to prevent one from reading the `stat` file of another user's process once it is forked, which can be done by manually changing the permissions of the file. However, this approach is not reliable because human are error-prone and whenever the step for altering permissions is inadvertently missed,

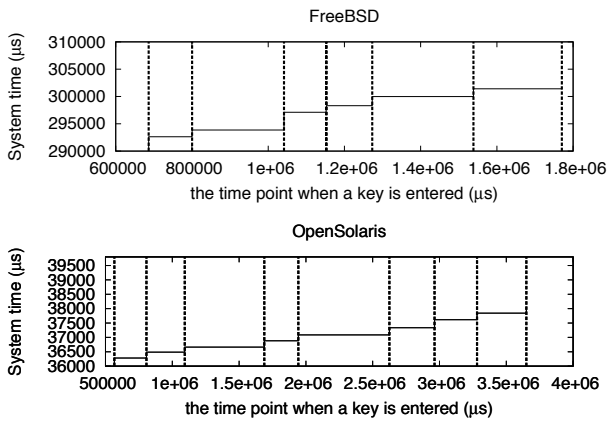

Figure 11: System time (solid line) vs. keystroke events (dashed line) in `vim` under FreeBSD (Release 7.1) and OpenSolaris (Release 2008.11). In the experiments, we found that the system time of `vim` changed only in response to keystrokes, which were recorded by shadow programs.

the door to our attack becomes wide open. The approach also affects the normal operations of common tools such as `ps` and `top`, which all depend on `stat` to acquire process information. A complete solution is to patch Linux kernels to remove the ESP and EIP information from a process's virtual file or move them into a separate file which can only be read by the owner. The problem is that there is no guarantee that other information disclosed by procfs will not lead to a similar attack (Section 6.1 and Section 6.2). Detection of our attack can also be hard, because our shadow process behaves exactly like the legitimate tools such as `top`, which also continuously read from virtual files. The shadow program can also hide its CPU usage by leveraging existing techniques [32]. Fundamentally, with the pervasiveness of multi-core systems that enable one process to effectively monitor another process's execution, we feel it is necessary to rethink the security implications of the public information available on current multi-user systems.

7 Related Work

It has long been known that individual users can be characterized by their unique and stable keystroke dynamics, the timing information that can be observed when one is typing [16]. Such information has been intensively studied for biometric authentication [21]. In comparison, little has been done to explore its potential for inferring the characters a user typed [6]. The first paper on this subject[13] proposes to measure inter-keystroke timings from the latencies between `SSH` packets [7] and use them to crack passwords. Our attack takes a different path to acquire timings: we take advantage of the information of a process exposed by procfs to find out when a key is received by the process, which has been made possible by the rapid development of multi-core techniques. Compared with the prior approach, our attack can happen to the clients who use a multi-user system locally as well as those who connect to the system remotely. Moreover, our timing analysis is much more accurate than the prior approach, through effective use of the information available from procfs. On the downside, we need a user account to launch our attack, which is not required by the prior approach. Another prior proposal measures CPU timings to acquire the information about the password a user enters [31]. This approach only gets the information such as password length and some special characters, and is subject to the interference of the activities such as processing mouse events, whereas our approach can accurately identify the events related to keystrokes and infer the characters being entered. Timing analysis has also been applied to attack cryptosystems [5, 34, 17, 8].

Keyboard acoustic emanations [34] also leak out information regarding a user's keystrokes. Such information has been leveraged by several prior approaches [2, 33, 3] to identify the keys being entered. Similar to our attack, some of these approaches also apply language models (including the high-order HMM) to infer English words. They all report very high success rates. Acoustic emanations are associated to individual keys, whereas timings are measured between a pair of keys. This makes character inference based on timings more challenging. On the other hand, acquisition of acoustic emanations requires physically implanting a recording device close to the victim, whereas our attack only needs a normal user account. Moreover, these attacks can only be used against a local user. In contrast, our approach works on both local and remote users.

8 Conclusion

In this paper, we present a new attack that allows a malicious user to eavesdrop on other users' keystrokes using procfs, a virtual file system that shares statistic information regarding individual users' processes. Our attack utilizes the stack information of a process present in its `stat` file on a Linux system to fingerprint its behavior when a keystroke is received. Such behavior is modeled as an ESP pattern of its system calls, which can be extracted from an application through automatic program analysis. During the runtime of the application, our approach shadows its process with another process to collect an ESP trace from its `stat` file. Our research shows that on a multi-core system, the shadow process can acquire a trace with a sufficient granularity for identifying keystroke events. This allows us to determine the tim-

ings between keystrokes and analyze them to infer the key sequence the victim entered. We also show that other information available from procfs can be of great help to character inference: knowing that the same user enters her password to the same application, we can combine multiple timing sequences related to the password to significantly reduce the space for searching it. We also propose to utilize the victim's writing style to infer the English words she enters. Both approaches are very effective, according to our experimental study.

Our attack can be further improved through adopting more advanced analysis techniques such as the high-order HMM and other language model. The same idea can also be applied to infer other user activities such as moving and clicking mouse, and even deduce others' secret keys. More generally, other information within procfs, such as system time, can be used for a similar attack, which threatens other UNIX-like systems such as FreeBSD and OpenSolaris. Research in these directions is left as our future work.

Acknowledgements

The authors thank our shepherd Angelos Stavrou for his guidance on the preparation of the final version, and anonymous reviewers for their comments on the draft of the paper. We also thank Rui Wang for his assistance in preparing one of the experiments reported in the paper. This work was supported in part by the National Science Foundation the Cyber Trust program under Grant No. CNS-0716292.

References

[1] Cryptography/frequency analysis. http://en.wikibooks.org/wiki/Cryptography:Frequency_analysis, Aug 2006.

[2] ASONOV, D., AND AGRAWAL, R. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy* (2004), pp. 3–11.

[3] BERGER, Y., WOOL, A., AND YEREDOR, A. Dictionary attacks using keyboard acoustics emanations. In *CCS* (2006), ACM, pp. 245–254.

[4] BERGROTH, L., HAKONEN, H., AND RAITA, T. A survey of longest common subsequence algorithms. In *Proceedings of Seventh International Symposium on String Processing and Information Retrieval* (2000), pp. 39–48.

[5] BRUMLEY, D., AND BONEH, D. Remote timing attacks are practical. In *In proceedings of the 12th Usenix Security Symposium* (2003).

[6] BUCHHOLTZ, M., GILMORE, S. T., HILLSTON, J., AND NIELSON, F. Securing statically-verified communications protocols against timing attacks. *Electronic Notes in Theoretical Computer Science 128*, 4 (2005), 123–143.

[7] DESIGNER, S., AND SONG, D. Passive analysis of ssh (secure shell) traffic. Openwall advisory OW-003, March 2001.

[8] DHEM, J. F., KOEUNE, F., LEROUX, P.-A., MESTRE, P., QUISQUATER, J.-J., AND WILLEMS, J.-L. A practical implementation of the timing attack. In *Proceedings of CARDIS* (1998), pp. 167–182.

[9] DISTROWATCH.COM. Top ten distributions: An overview of today's top distributions. http://distrowatch.com/dwres.php?resource=major, 2008.

[10] EDIT VIRTUAL LANGUAGE CENTER. Word frequency lists. http://www.edict.com.hk/textanalyser/wordlists.htm, as of September, 2008.

[11] FERRELL, J. procfs: Gone but not forgotten. http://www.freebsd.org/doc/en/articles/linux-users/procfs.html, 2009.

[12] FRANCOIS, M. J., AND PAUL, H. J. Automatic word recognition based on second-order hidden markov models. In *ICSLP* (1994), pp. 247–250.

[13] HOGYE, M. A., HUGHES, C. T., SARFATY, J. M., AND WOLF, J. D. Analysis of the feasibility of keystroke timing attacks over ssh connections. Technical Report CS588, School of Engineering and Applied Science, University of Virginia, December 2001.

[14] INC., R. Process directories. http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/en-US/ReferenceGuide/s2-proc-processdirs.html, 2007.

[15] JONES, N. C., AND PEVZNER, P. A. *An Introduction to Bioinformatics Algorithms*. the MIT Press, August 2004.

[16] JOYCE, R., AND GUPTA, G. Identity authorization based on keystroke latencies. *Communications of the ACM 33*, 2 (1990), 168–176.

[17] KOCHER, P., JAE, J., AND JUN, B. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology* (1999), Springer-Verlag, pp. 388–397.

[18] LEECH, G., RAYSON, P., AND WILSON, A. Word frequencies in written and spoken english: based on the british national corpus. http://www.comp.lancs.ac.uk/ucrel/bncfreq.

[19] LOSCOCCO, P., AND SMALLEY, S. procfs analysis. http://www.nsa.gov/SeLinux/papers/slinux/node57.html, February 2001.

[20] LUK, C. K., COHN, R., MUTH, R., PATIL, H., KLAUSER, A., LOWNEY, G., WALLACE, S., REDDI, V. J., AND HAZELWOOD, K. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation* (2005), pp. 190–200.

[21] MONROSE, F., AND RUBIN, A. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM conference on Computer and communications security* (1997), ACM Press, pp. 48–56.

[22] PETERSSON, J. What is linux-gate.so.1? http://www.trilithium.com/johan/2005/08/linux-gate/, as of September, 2008.

[23] PROVOS, N. Systrace - interactive policy generation for system calls. http://www.citi.umich.edu/u/provos/systrace/, 2006.

[24] RABINER, L. R. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*, 2 (1989), 257–286.

[25] SECURITY, S. C. Timing analysis is not a real-life threat to ssh secure shell users. http://www.ssh.com/company/news/2001/english/all/article/204/, November 2001.

[26] SONG, D. X., WAGNER, D., AND TIAN, X. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium* (2001), USENIX Association.

[27] SOURCEFORGE.NET. `http://sourceforge.net/projects/strace/`, August 2008.

[28] TEAM, G. `http://www.gtk.org`, as of September, 2008.

[29] TEAM, P. Pax address space layout randomization (aslr). `http://pax.grsecurity.net/docs/aslr.txt`, March 2003.

[30] TEAM, P. `http://pax.grsecurity.net/`, as of September, 2008.

[31] TROSTLE, J. Timing attacks against trusted path. In *IEEE Symposium on Security and Privacy* (1998).

[32] TSAFRIR, D., ETSION, Y., AND FEITELSON, D. G. Secretly monopolizing the cpu without superuser privileges. In *Proceedings of 16th USENIX Security Symposium* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–18.

[33] ZHANG, L., ZHOU, F., AND TYGAR, J. D. Keyboard acoustic emanations revisited. In *CCS'05: ACM Conference on Computer and Communications Security* (2005), ACM Press, pp. 373–382.

[34] ZHOU, Y., AND FENG, D. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. csrc.nist.gov/groups/STM/cmvp/documents/fips140-3/physec/papers/physecpaper19.pdf, December 2005.

Notes

[1]The program is actually a simplified version of `vim`.

[2]Some old Linux distributions such as RedHat Enterprise 4 do not use vDSO, and instead then entry of their system calls points to `_dl_sysinfo_int80` in library /lib/ld-linux.so or /lib/ld.so.

[3]We designed our attack in a way that a keystroke event can be reliably identified even in the presence of some missing ESP/EIP values, which could happen when the shadow process is preempted by other processes (Section 3).

[4]After the application enter the state that keystroke inputs are expected, our approach waits for a time period before exporting the first sequence. This allows for the accomplishment of all the system calls prior to keystrokes. Similarly, the second sequence is not exported until the keystroke happens for a while so as to ensure that all the system calls related to the stroke are completed.

[5]There are actually two events associated with a keystroke: key press and key release. We use the first event here for the simplicity of explanation. Our technique can actually be applied to both events.

[6]We did not use the instructions such as '`ret`' to identify the end of a call-back function because compiler optimization could remove such instructions from a binary executable.

[7]Some Linux versions such as RedHat [14] turn off the permissions on `maps` but `stat` is always open.

[8]Theoretically, this approach may not eliminate false positives when it comes to non-deterministic applications, because these applications may contain ESP sequences we did not observe during the offline analysis.

[9]The prior work used 10 letters and 5 numbers. We increased the number of letter keys to get a larger set of legitimate words for our experiment on English text.

[10]The factor is actually below what was reported in the prior work [26]. A possibility is that we adopted 225 key pairs rather than 142 used in the prior work.

[11]We did not choose longer words in our experiment to avoid intensive computation. However, such a word can also be learnt through splitting it into shorter segments and analyzing them using different HMMs.

[12]It is reported that FreeBSD moves to phase out procfs [11].

[13]The possibility of timing attack on `SSH` has also been briefly discussed in [26].

A Practical Congestion Attack on Tor Using Long Paths

Nathan S. Evans
Colorado Research Institute for Security and Privacy University of Denver
Email: nevans6@du.edu

Roger Dingledine
The Tor Project
Email: arma@mit.edu

Christian Grothoff
Colorado Research Institute for Security and Privacy University of Denver
Email: christian@grothoff.org

Abstract

In 2005, Murdoch and Danezis demonstrated the first practical congestion attack against a deployed anonymity network. They could identify which relays were on a target Tor user's path by building paths one at a time through every Tor relay and introducing congestion. However, the original attack was performed on only 13 Tor relays on the nascent and lightly loaded Tor network.

We show that the attack from their paper is no longer practical on today's 1500-relay heavily loaded Tor network. The attack doesn't scale because a) the attacker needs a tremendous amount of bandwidth to measure enough relays during the attack window, and b) there are too many false positives now that many other users are adding congestion at the same time as the attacks.

We then strengthen the original congestion attack by combining it with a novel bandwidth amplification attack based on a flaw in the Tor design that lets us build long circuits that loop back on themselves. We show that this new combination attack is practical and effective by demonstrating a working attack on today's deployed Tor network. By coming up with a model to better understand Tor's routing behavior under congestion, we further provide a statistical analysis characterizing how effective our attack is in each case.

1 Introduction

This paper presents an attack which exploits a weakness in Tor's circuit construction protocol to implement an improved variant of Murdoch and Danezis's congestion attack [26, 27]. Tor [12] is an anonymizing peer-to-peer network that provides users with the ability to establish low-latency TCP tunnels, called circuits, through a network of relays provided by the peers in the network. In 2005, Murdoch and Danezis were able to determine the path that messages take through the Tor network by causing congestion in the network and then observing the changes in the traffic patterns.

While Murdoch and Danezis's work popularized the idea proposed in [1] of an adversary perturbing traffic patterns of a low-latency network to deanonymize its users, the original attack no longer works on the modern Tor network. In a network with thousands of relays, too many relays share similar latency characteristics and the amount of congestion that was detectable in 2005 is no longer significant; thus, the traffic of a single normal user does not leave an easily distinguishable signature in the significantly larger volume of data routed by today's Tor network.

We address the original attack's weaknesses by combining JavaScript injection with a selective and asymmetric denial-of-service (DoS) attack to obtain specific information about the path selected by the victim. As a result, we are able to identify the entire path for a user of today's Tor network. Because our attack magnifies the congestion effects of the original attack, it requires little bandwidth on the part of the attacker. We also provide an improved method for evaluating the statistical significance of the obtained data, based on Tor's message scheduling algorithm. As a result, we are not only able to determine which relays make up the circuit with high probability, we can also quantify the extent to which the attack succeeds. This paper presents the attack and experimental results obtained from the actual Tor network.

We propose some non-trivial modifications to the current Tor protocol and implementation which would raise the cost of the attack. However, we emphasize that a full defense against our attack is still not known.

Just as Murdoch and Danezis's work applied to other systems such as MorphMix [24] or Tarzan [36], our improved attack and suggested partial defense can also be generalized to other networks using onion routing. Also, in contrast to previously proposed solutions to congestion attacks [18,22–24,28,30,35,36], our proposed modifications do not impact the performance of the anonymizing network.

2 Related Work

Chaum's mixes [3] are a common method for achieving anonymity. Multiple encrypted messages are sent to a mix from different sources and each is forwarded by the mix to its respective destination. Combinations of artificial delays, changes in message order, message batching, uniform message formats (after encryption), and chaining of multiple mixes are used to further mask the correspondence between input and output flows in various variations of the design [5, 7, 8, 17, 21, 25, 32, 33]. Onion routing [16] is essentially the process of using an initiator-selected chain of low-latency mixes for the transmission of encrypted streams of messages in such a way that each mix only knows the previous and the next mix in the chain, thus providing initiator-anonymity even if some of the mixes are controlled by the adversary.

2.1 Tor

Tor [12] is a distributed anonymizing network that uses onion routing to provide anonymity for its users. Most Tor users access the Tor network via a local proxy program such as Privoxy [20] to tunnel the HTTP requests of their browser through the Tor network. The goal is to make it difficult for web servers to ascertain the IP address of the browsing user. Tor provides anonymity by utilizing a large number of distributed volunteer-run relays (or routers). The Tor client software retrieves a list of participating relays, randomly chooses some number of them, and creates a circuit (a chain of relays) through the network. The circuit setup involves establishing a session key with each router in the circuit, so that data sent can be encrypted in multiple layers that are peeled off as the data travels through the network. The client encrypts the data once for each relay, and then sends it to the first relay in the circuit; each relay successively peels off one encryption layer and forwards the traffic to the next link in the chain until it reaches the final node, the exit router of the circuit, which sends the traffic out to the destination on the Internet.

Data that passes through the Tor network is packaged into fixed-sized cells, which are queued upon receipt for processing and forwarding. For each circuit that a Tor router is a part of, the router maintains a separate queue and processes these queues in a round-robin fashion. If a queue for a circuit is empty it is skipped. Other than using this fairness scheme, Tor does not intentionally introduce any latency when forwarding cells.

The Tor threat model differs from the usual model for anonymity schemes [12]. The traditional threat model is that of a global passive adversary: one that can observe all traffic on the network between any two links. In contrast, Tor assumes a non-global adversary which can only observe some subset of the connections and can control only a subset of Tor nodes. Well-known attack strategies such as blending attacks [34] require more powerful attackers than those permitted by Tor's attacker model. Tor's model is still valuable, as the resulting design achieves a level of anonymity that is sufficient for many users while providing reasonable performance. Unlike the aforementioned strategies, the adversary used in this paper operates within the limits set by Tor's attacker model. Specifically, our adversary is simply able to run a Tor exit node and access the Tor network with resources similar to those of a normal Tor user.

2.2 Attacks on Tor and other Mixes

Many different attacks on low-latency mix networks and other anonymization schemes exist, and a fair number of these are specifically aimed at the Tor network. These attacks can be broadly grouped into three categories: path selection attacks, passive attacks, and active attacks. Path selection attacks attempt to invalidate the assumption that selecting relays at random will usually result in a safe circuit. Passive attacks are those where the adversary in large part simply observes the network in order to reduce the anonymity of users. Active attacks are those where the adversary uses its resources to modify the behavior of the network; we'll focus here on a class of active attacks known as congestion or interference attacks.

2.2.1 Path Selection Attacks

Path selection is crucial for the security of Tor users; in order to retain anonymity, the initiator needs to choose a path such that the first and last relay in the circuit won't collude. By selecting relays at random during circuit creation, it could be assumed that the probability of finding at least one non-malicious relay would increase with longer paths. However, this reasoning ignores the possibility that malicious Tor routers might choose only to facilitate connections with other adversary-controlled relays and discard all other connections [2]; thus the initiator either constructs a fully malicious circuit upon randomly selecting a malicious node, or fails that circuit and tries again. This type of attack suggests that longer circuits do not guarantee stronger anonymity.

A variant of this attack called "packet spinning" [30] attempts to force users to select malicious routers by causing legitimate routers to time out. Here the attacker builds circular paths throughout the Tor network and transmits large amounts of data through those paths in order to keep legitimate relays busy. The attacker then runs another set of (malicious) servers which would eventually be selected by users because of the attacker-generated load on all legitimate mixes. The attack is successful if, as a result, the initiator chooses only malicious servers for its circuit, making deanonymization trivial.

2.2.2 Passive Attacks

Several passive attacks on mix systems were proposed by Back et al. [1]. The first of these attacks is a "packet counting" attack, where a global passive adversary simply monitors the initiator's output to discover the number of packets sent to the first mix, then observes the first mix to watch for the same number of packets going to some other destination. In this way, a global passive adversary could correlate traffic to a specific user. As described by Levine et al. [23], the main method of defeating such attacks is to pad the links between mixes with cover traffic. This defense is costly and may not solve the problem when faced with an active attacker with significant resources; an adversary with enough bandwidth can deal with cover traffic by using up as much of the allotted traffic between two nodes as possible with adversary-generated traffic [4]. As a result, no remaining bandwidth is available for legitimate cover traffic and the adversary can still deduce the amount of legitimate traffic that is being processed by the mix. This attack (as well as others described in this context) requires the adversary to have significant bandwidth. It should be noted that in contrast, the adversary described by our attack requires only the resources of an average mix operator.

Low-latency anonymity systems are also vulnerable to more active timing analysis variations. The attack presented in [23] is based on an adversary's ability to track specific data through the network by making minor timing modifications to it. The attack assumes that the adversary controls the first and last nodes in the path through the network, with the goal of discovering which destination the initiator is communicating with. The authors discuss both correlating traffic "as is" as well as altering the traffic pattern at the first node in order to make correlation easier at the last node. For this second correlation attack, they describe a packet dropping technique which creates holes in the traffic; these holes then percolate through the network to the last router in the path. The analysis showed that without cover traffic (as employed in Tarzan [14, 15]) or defensive dropping [23], it is relatively easy to correlate communications through mix networks. Even with "normal" cover traffic where all packets between nodes look the same, Shmatikov and Wang show that the traffic analysis attacks are still viable [35]. Their proposed solution is to add cover traffic that mimics traffic flows from the initiator's application.

A major limitation of all of the attacks described so far is that while they work well for small networks, they do not scale and may fail to produce reliable results for larger anonymizing networks. For example, Back's active latency measuring attack [1] describes measuring the latencies of circuits and then trying to determine the nodes that were being utilized from the latency of a specific circuit. As the number of nodes grows, this attack becomes more difficult (due to an increased number of possible circuits), especially as more and more circuits have similar latencies.

2.2.3 Congestion Attacks

A more powerful relative of the described timing attacks is the clogging or congestion attack. In a clogging attack, the adversary not only monitors the connection between two nodes but also creates paths through other nodes and tries to use all of their available capacity [1]; if one of the nodes in the target path is clogged by the attacker, the observed speed of the victim's connection should change.

In 2005, Murdoch and Danezis described an attack on Tor [27] in which they could reveal all of the routers involved in a Tor circuit. They achieved this result using a combination of a circuit clogging attack and timing analysis. By measuring the load of each node in the network and then subsequently congesting nodes, they were able to discover which nodes were participating in a particular circuit. This result is significant, as it reduces Tor's security during a successful attack to that of a collection of one hop proxies. This particular attack worked well on the fledgling Tor network with approximately fifty nodes; the authors experienced a high success rate and no false positives. However, their clogging attack no longer produces a signal that stands out on the current Tor network with thousands of nodes. Because today's Tor network is more heavily used, circuits are created and destroyed more frequently, so the addition of a single clogging circuit has less impact. Also, the increased traffic transmitted through the routers leads to false positives or false negatives due to normal network fluctuations. We provide details about our attempt to reproduce Murdoch and Danezis's work in Section 6.

McLachlan and Hopper [24] propose a similar circuit clogging attack against MorphMix [33], disproving claims made in [36] that MorphMix is invulnerable to such an attack. Because all MorphMix users are *required* to also be mix servers, McLachlan and Hopper achieve a stronger result than Murdoch and Danezis: they can identify not only the circuit, but the user as well.

Hopper et al. [19] build on the original clogging attack idea to construct a network latency attack to guess the location of Tor users. Their attack is two-phase: first use a congestion attack to identify the relays in the circuit, and then build a parallel circuit through those relays to estimate the latency between the victim and the first relay. A key contribution from their work is a more mathematical approach that quantifies the amount of information leaked in bits over time. We also note that without a working congestion attack, the practicality of their overall approach is limited.

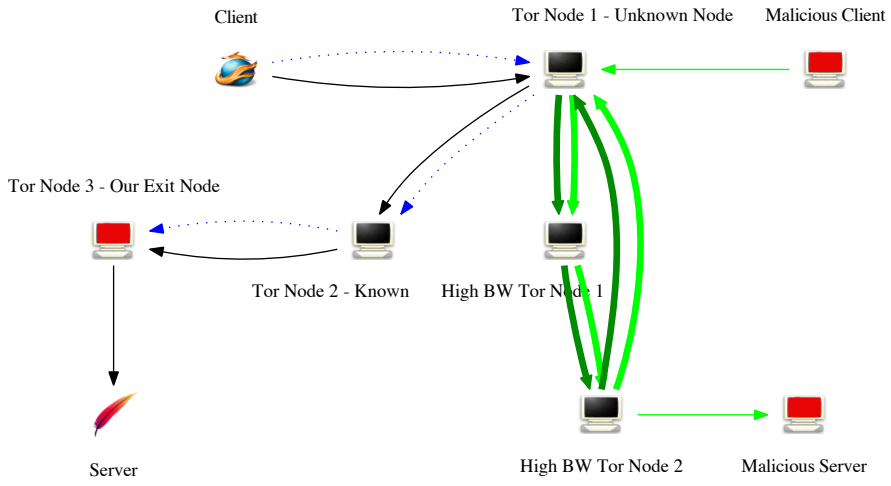

Figure 1: Attack setup. This figure illustrates the normal circuit constructed by the victim to the malicious Tor exit node and the "long" circuit constructed by the attacker to congest the entry (or guard) node used by the victim. The normal thin line from the *client* node to the *server* represents the victim circuit through the Tor network. The unwitting client has chosen the exit server controlled by the adversary, which allows the JavaScript injection. The double thick lines represent the long circular route created by the *malicious client* through the first Tor router chosen by the client. The dotted line shows the path that the JavaScript pings travel.

3 Our Attack

Three features of Tor's design are crucial for our attack. First of all, Tor routers do not introduce any artificial delays when routing requests. As a result, it is easy for an adversary to observe changes in request latency. Second, the addresses of all Tor routers are publicly known and easily obtained from the directory servers. Tor developers are working on extensions to Tor (called bridge nodes [10, 11]) that would invalidate this assumption, but this service was not widely used at the time of this writing. Finally, the latest Tor server implementation that was available at the time we concluded our original attacks (Tor version 0.2.0.29-rc) did not restrict users from establishing paths of arbitrary length, meaning that there was no restriction in place to limit constructing long paths through Tor servers.[1] We used a modified client version (based on 0.2.0.22-rc) which used a small fixed path length (specifically three) but modified it to use a variable path length specified by our attacker.

Fig. 1 illustrates the three main steps of our attack. First, the adversary needs to ensure that the initiator repeatedly performs requests at known intervals. Second, the adversary observes the pattern in arrival times of these requests. Finally, the adversary changes the pattern by selectively performing a novel clogging attack on Tor routers to determine the entry node. We will now describe each of these steps in more detail.

3.1 JavaScript Injection

Our attack assumes that the adversary controls an exit node which is used by the victim to access an HTTP server. The attacker uses the Tor exit node to inject a small piece of JavaScript code (shown in Fig. 2) into an HTML response. It should be noted that most Tor users do **not** disable JavaScript and that the popular Tor Button plugin [31] and Privoxy [20] also do not disable JavaScript code; doing so would prevent Tor users from accessing too many web pages. The JavaScript code causes the browser to perform an HTTP request every second, and in response to each request, the adversary uses the exit node to return an empty response, which is thrown away by the browser. Since the JavaScript code may not be able to issue requests precisely every second, it also transmits the local system time (in milliseconds) as part of the request. This allows the adversary to determine the time difference between requests performed by the browser with sufficient precision. (Clock skew on the systems of the adversary and the victim is usually insignificant for the duration of the attack.) While JavaScript is not the only conceivable way for an attacker to cause a browser to transmit data at regular intervals (alternatives include HTTP headers like `refresh` [13]

[1] Tor version 0.2.1.3-alpha and later servers restrict path lengths to a maximum of eight because of this work.

```
<script language="javascript">
 var count,timer,xmlhttp = 0;
 function runonce() {
  xmlhttp = new XMLHttpRequest();   }
 function start() {
  xmlhttp.abort();
  xmlhttp = new XMLHttpRequest();
  count++;
  if (timer) clearTimeout(timer);
  timer = setTimeout("start()", 1000);
  myDate = new Date();
  xmlhttp.open("GET",
    "/reportIn.html?num=" + count +
    "&time=" + myDate.getTime(),true);
  xmlhttp.send("");
 }
</script>
```

Figure 2: JavaScript code injected by the adversary's exit node. Note that other techniques such as HTML refresh, could also be used to cause the browser to perform periodic requests.

and HTML images [19]), JavaScript provides an easy and generally rather dependable method to generate such a signal.

The adversary then captures the arrival times of the periodic requests performed by the browser. Since the requests are small, an idle Tor network would result in the differences in arrival times being roughly the same as the departure time differences — these are known because they were added by the JavaScript as parameters to the requests. Our experiments suggest that this is often true for the real network, as most routers are not seriously congested most of the time. This is most likely in part due to TCP's congestion control and Tor's built-in load balancing features. Specifically, the variance in latency between the periodic HTTP requests without an active congestion attack is typically in the range of 0–5 s.

However, the current Tor network is usually not entirely idle and making the assumption that the victim's circuit is idle is thus not acceptable. Observing congestion on a circuit is not enough to establish that the node under the congestion attack by the adversary is part of the circuit; the circuit may be congested for other reasons. Hence, the adversary needs to also establish a baseline for the congestion of the circuit without an active congestion attack. Establishing measurements for the baseline is done before and after causing congestion in order to ensure that observed changes during the attack are caused by the congestion attack and not due to unrelated changes in network characteristics.

The attacker can repeatedly perform interleaved measurements of both the baseline congestion of the circuit and the congestion of the circuit while attacking a node presumed to be on the circuit. The attacker can continue the measurements until either the victim stops using the circuit or until the mathematical analysis yields a node with a substantially higher deviation from the baseline under congestion compared to all other nodes. Before we can describe details of the mathematical analysis, however, we have to discuss how congestion is expected to impact the latency measurements.

3.2 Impact of Congestion on Arrival Times

In order to understand how the congestion attack is expected to impact latency measurements, we first need to take a closer look at how Tor schedules data for routing. Tor makes routing decisions on the level of fixed-size *cells*, each containing 512 bytes of data. Each Tor node routes cells by going round-robin through the list of all circuits, transmitting one packet from each circuit with pending data (see Fig. 3). Usually the number of (active) circuits is small, resulting in little to no delay. If the number of busy circuits is large, messages may start to experience significant delays as the Tor router iterates over the list (see Fig. 4).

Since the HTTP requests transmitted by the injected JavaScript code are small ($\sim$250 bytes, depending on count and time), more than one request can fit into a single Tor cell. As a result multiple of these requests will be transmitted at the same time if there is congestion at a router. A possible improvement to our attack would be to use a lower level API to send the packets, as the XMLHttpRequest object inserts unnecessary headers into the request/response objects.

We will now characterize the network's behavior under congestion with respect to request arrival times. Assuming that the browser transmits requests at a perfectly steady rate of one request per second, a congested router introducing a delay of (at most) n seconds would cause groups of n HTTP requests to arrive with delays of approximately $0, 1, \ldots, n-1$ seconds respectively: the first cell is delayed by $n-1$ seconds, the cell arriving a second later by $n-2$ seconds, and the n-th cell arrives just before the round-robin scheduler processes the circuit and sends all n requests in one batch. This characterization is of course a slight idealization in that it assumes that n is small enough to allow all of the HTTP requests to be grouped into one Tor cell and that there are no other significant fluctuations. Furthermore, it assumes that the amount of congestion caused by the attacker is perfectly steady for the duration of the time measurements, which may not be the case. However, even without these idealizations it is easy to see that the resulting latency histograms would still become "flat" (just not as perfectly

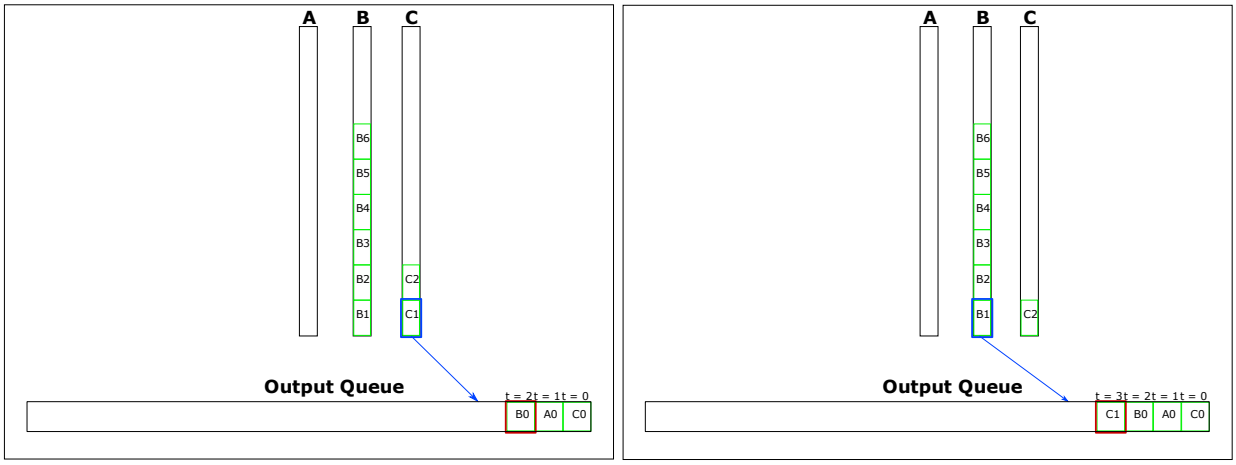

Figure 3: This example illustrates a Tor router which currently is handling three circuits at two points in time ($t = 3$ and $t = 4$). Circuits (A, B and C) have queues; cells are processed one at a time in a round-robin fashion. As the number of circuits increases, the time to iterate over the queues increases. The left figure shows the circuit queues and output queue before selection of cell `C1` for output and the right figure shows the queues after queueing `C1` for output. The thicker bottom box of queue C (left) and queue B (right) shows the current position of the round-robin queue iterator. At time $t = 1$ the last cell from queue A was processed leaving the queue A empty. As a result, queue A is skipped after processing queue C.

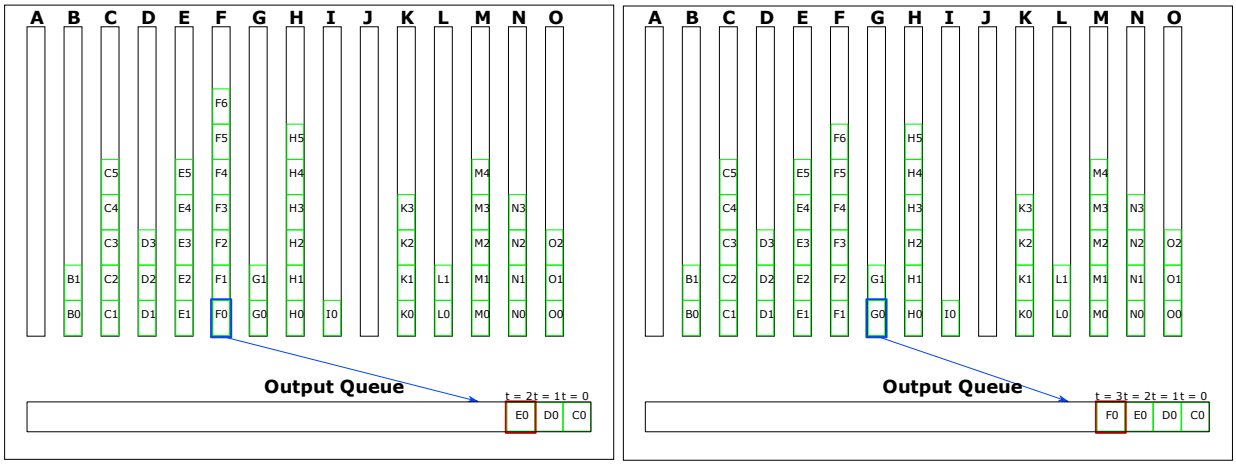

Figure 4: This example illustrates a Tor router under congestion attack handling 15 circuit queues. Note that if a circuit includes a node multiple times, the node assigns the circuit multiple circuit queues. In this example, not all of the circuit queues are busy — this may be because the circuits are not in use or because other routers on the circuit are congested. As in Fig. 3, the left and right figures show the state of the mix before and after queueing a cell, in this case `F0`.

regular in terms of arrival patterns) assuming the load caused by the attacker is sufficiently high.

Since we ideally expect delays in message arrival times for a congested circuit to follow a roughly flat distribution between zero and n, it makes sense to compute a histogram of the delays in message arrival times. If the congestion attack is targeting a node on the circuit, we would expect to see a roughly equal number of messages in each interval of the histogram. We will call the shape of the resulting histogram *horizontal*. If the circuit is not congested, we expect to see most messages arrive without significant delay which would place them in the bucket for the lowest latency. We will call the shape of the resulting histogram *vertical*. So for example, in Fig. 6 the control data are vertical, whereas the attack data are more horizontal.

Note that the clock difference between the victim's system and the adversary as well as the minimal network delay are easily eliminated by normalizing the observed time differences. As a result, the latency histograms should use the increases in latency over the smallest observed latency, not absolute latencies.

3.3 Statistical Evaluation

In order to numerically capture congestion at nodes we first measure the node's *baseline* latency, that is latency without an active congestion attack (at least as far as we know). We then use the observed latencies to create n bins of latency intervals such that each bin contains the same number of data points. Using the χ^2-test we could then determine if the latency pattern at the respective peer has changed "significantly". However, this simplistic test is insufficient. Due to the high level of normal user activity, nodes frequently do change their behavior in terms of latencies, either by becoming congested or by congestion easing due to clients switching to other circuits. For the attacker, congestion easing (the latency histogram getting more vertical) is exactly the opposite of the desired effect. Hence the ordinary χ^2 test should not be applied without modification. What the attacker is looking for is the histogram becoming more horizontal, which for the distribution of the bins means that there are fewer values in the low-latency bins and more values in the high-latency bins. For the medium-latency bins no significant change is expected (and any change there is most likely noise).

Hence we modify our computation of the χ^2 value such that we only include changes in the anticipated direction: for the bins corresponding to the lowest third of the latencies, the square of the difference between expected and observed number of events is only counted in the summation if the number of observed events is lower than expected. For the bins corresponding to the highest third of the latencies, the square of the difference between expected and observed number of events is only counted if the number of observed events is higher than expected. Since changes to the bins in the middle third are most likely noise, those bins are not included in the χ^2 calculation at all (except as a single additional degree of freedom).

Using this method, a single iteration of measuring the baseline and then determining that there was a significant increase in latency (evidenced by a large χ^2-value), only signifies that congestion at the guard for the victim circuit was correlated (in time) with the congestion caused by the attacker. Naturally, correlation does not imply causality; in fact, for short (30–60 s) attack runs it frequently happens that the observed χ^2-value is higher for some false-positive node than when attacking the correct guard node. However, such accidental correlations virtually never survive iterated measurements of the latency baseline and χ^2-values under attack.

3.4 Congestion Attack

Now we focus on how the attacker controlling the exit node of the circuit will cause significant congestion at nodes that are suspected to be part of the circuit. In general, we will assume that all Tor routers are suspects and that in the simplest case, the attacker will iterate over all known Tor routers with the goal of finding which of these routers is the entry point of the circuit.

For each router X, the attacker constructs a long circuit that repeatedly includes X on the path. Since Tor relays will tear down a circuit that tries to extend to the previous node, we have to use two (or more) other (preferably high-bandwidth) Tor routers before looping back to X. Note that the attacker could choose two different (involuntary) helper nodes in each loop involving X. Since X does not know that the circuit has looped back to X, Tor will treat the long attack circuit as many different circuits when it comes to packet scheduling (Fig. 4).

Once the circuit is sufficiently long (we typically found 24 hops to be effective, but in general this depends on the amount of congestion established during the baseline measurements), the attacker uses the circuit to transmit data. Note that a circuit of length m would allow an attacker with p bandwidth to consume $m \cdot p$ bandwidth on the Tor network, with X routing as much as $\frac{m \cdot p}{3}$ bandwidth. Since X now has to iterate over an additional $\frac{m}{3}$ circuits, this allows the attacker to introduce large delays at this specific router. The main limitation for the attacker here is time. The larger the desired delay d and the smaller the available attacker bandwidth p the longer it will take to construct an attack circuit of sufficient length m: the number of times that the victim node is part of the attack circuit is proportional to the length of

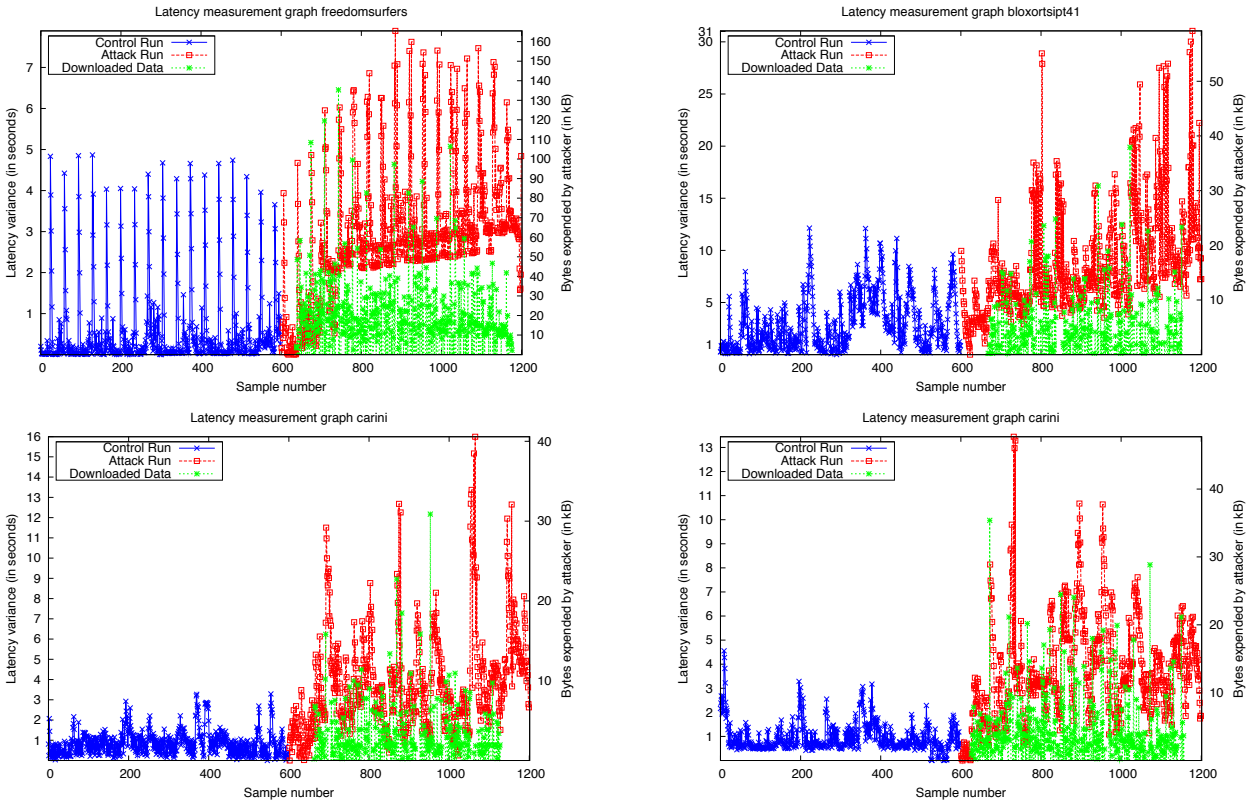

Figure 5: These figures show the results of perturbation of circuits in Tor and the resulting effects on latency. The x-axes show sample numbers (one per second), and the (left) y-axes are latency variance observed on the circuits in seconds. The attack on the first router of each circuit starts at time 600; the third line shows the amount of data (scaled) that transferred through the attack circuit (scaled to the right y-axes). These are individual trials; each shows a single control run and a single attack run.

the circuit m. In other words, the relationship between p, m and the delay d is $d \sim p \cdot m$.

If the router X is independent of the victim circuit, the measured delays should not change significantly when the attack is running. If X is the entry node, the attacker should observe a delay pattern that matches the power of the attack – resulting in a horizontal latency variance histogram as described in Section 3.2. The attacker can vary the strength of the attack (or just switch the long attack circuit between idle and busy a few times) to confirm that the victim's circuit latency changes correlate with the attack. It should be noted that the attacker should be careful to not make the congestion attack too powerful, especially for low-bandwidth targets. In our experiments we sometimes knocked out routers (for a while) by giving them far too much traffic. As a result, instead of receiving requests from the JavaScript code with increasing latencies, the attacker suddenly no longer receives requests at all, which gives no useful data for the statistical evaluation.

3.5 Optimizations

The adversary can establish many long circuits to be used for attacks before trying to deanonymize a particular victim. Since idle circuits would not have any impact on measuring the baseline (or the impact of using another attack circuit), this technique allows an adversary to eliminate the time needed to establish circuits. As users can only be expected to run their browser for a few minutes, eliminating this delay may be important in practice; even users that may use their browser for hours are likely to change between pages (which might cause Tor to change exit nodes) or disable Tor.

In order to further speed up the process, an adversary can try to perform a binary search for X by initially running attacks on half of the routers in the Tor network. With pre-built attack circuits adding an almost unbounded multiplier to the adversary's resources, it is conceivable that a sophisticated attacker could probe a network of size s in $\log_2 s$ rounds of attacks.

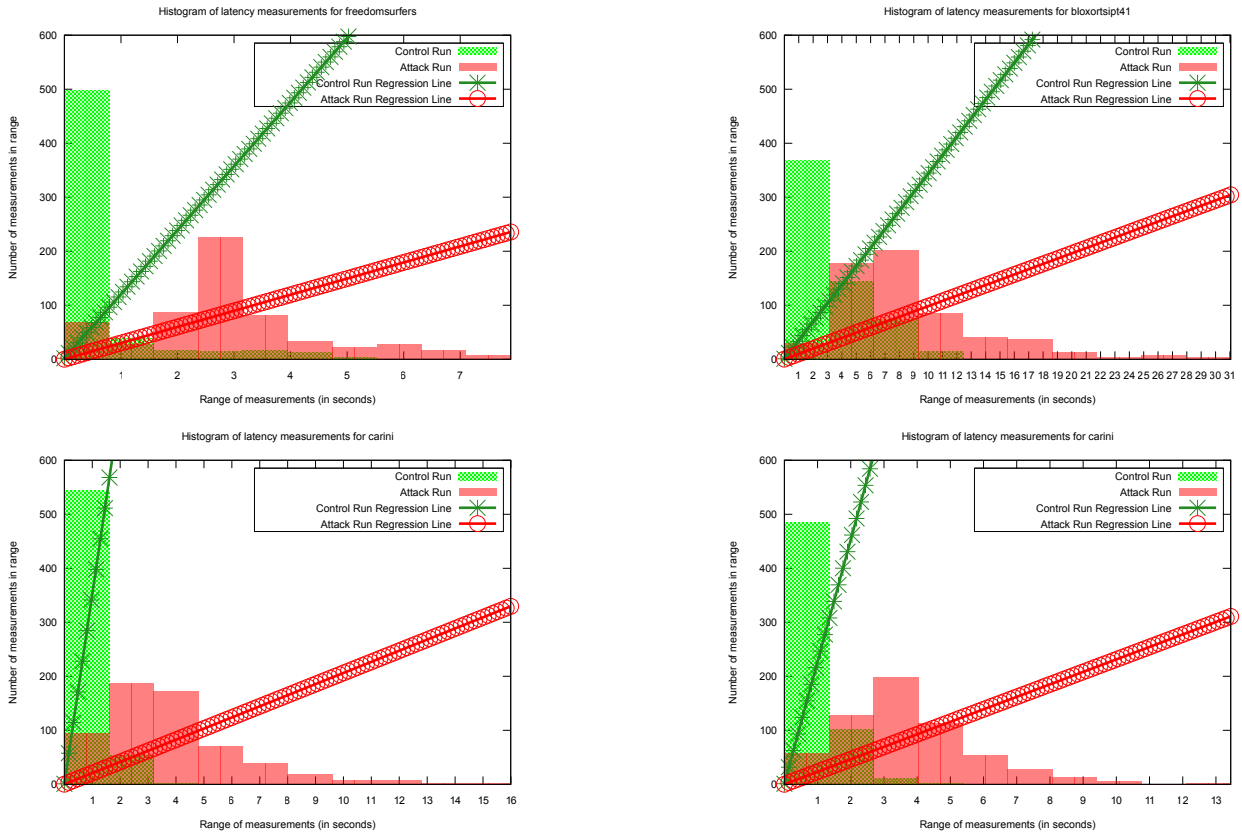

Figure 6: These figures show the results of four independent runs of our congestion attack. In the histograms the x-axis groups ranges of latency variance values together and the y-axis represents the number of readings received in that range. The hash marked bars represent the unperturbed measurements on a circuit and the plain bars show measurements from the same circuit during the attack. The effect of the attack is a shift to higher latency values. The first and second lines are linear least squares fit approximations for the baseline and congestion runs, respectively. These data show the difference between a single control/attack run and are not averages of many runs.

In practice, pre-building a single circuit that would cause congestion for half the network is not feasible; the Tor network is not stable enough to sustain circuits that are thousands of hops long. Furthermore, the differences in available bandwidth between the routers complicates the path selection process. In practice, an adversary would most likely pre-build many circuits of moderate size, forgoing some theoretical bandwidth and attack duration reductions for circuits that are more reliable. Furthermore, the adversary may be able to exclude certain Tor routers from the set of candidates for the first hop based on the overall round-trip latency of the victim's circuit. The Tor network allows the adversary to measure the latency between any two Tor routers [19, 27]; if the overall latency of the victim's circuit is smaller than the latency between the known second router on the path and another router Y, then Y is most likely not a candidate for the entry point.

Finally, the adversary needs to take into consideration that by default, a Tor user switches circuits every 10 minutes. This further limits the window of opportunity for the attacker. However, depending on the browser, the adversary may be able to cause the browser to pipeline HTTP requests which would not allow Tor to switch circuits (since the HTTP session would not end). Tor's circuit switching also has advantages for the adversary: every 10 minutes there is a new chance that the adversary-controlled exit node is chosen by a particular victim. Since users only use a small number of nodes for the first node on a circuit (these nodes are called guard nodes [29]), the adversary has a reasonable chance over time to determine these guard nodes. Compromising one of the guard nodes would then allow full deanonymization of the target user.

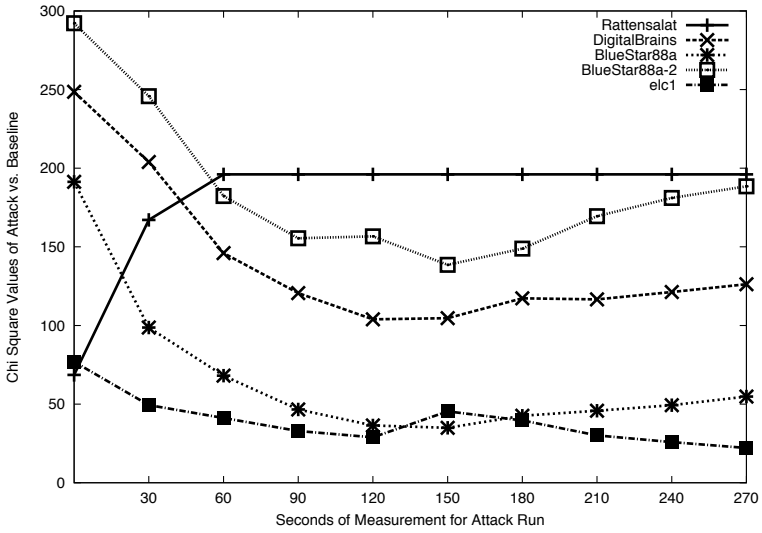

Figure 7: This figure shows the development of χ^2 values (using the modified χ^2 calculation as described in Section 3.3) for the various candidates over a prolonged period of performing a congestion attack on the various nodes. The χ^2 values are computed against a five-minute baseline obtained just prior to the congestion attack. The χ^2 value of the correct entry node quickly rises to the top whereas the χ^2 values for all of the other candidates are typically lower after about a minute of gathering latency information under congestion. This illustrates that a few minutes are typically sufficient to obtain a meaningful χ^2 value.

4 Experimental Results

The results for this paper were obtained by attacking Tor routers on the real, deployed Tor network (initial measurements were done during the Spring and Summer of 2008; additional data was gathered in Spring 2009 with an insignificantly modified attacker setup; the modifications were needed because our original attack client was too outdated to work with the majority of Tor routers at the time). In order to confirm the accuracy of our experiments and avoid ethical problems, we did not attempt to deanonymize real users. Instead, we established our own client circuits through the Tor network to our malicious exit node and then confirmed that our statistical analysis was able to determine the entry node used by our own client. Both the entry nodes and the second nodes on the circuits were normal nodes in the Tor network outside of our control.

The various roles associated with the adversary (exit node, malicious circuit client, and malicious circuit webserver) as well as the "deanonymized" victim were distributed across different machines in order to minimize interference between the attacking systems and the targeted systems. For the measurements we had the simulated victim running a browser requesting and executing the malicious JavaScript code, as well as a machine running the listening server to which the client transmits the "ping" signal approximately every second (Fig. 1). The browser always connected to the same unmodified Tor client via Privoxy [20]. The Tor client used the standard configuration except that we configured it to use our malicious exit node for its circuits. The other two nodes in the circuit were chosen at random by Tor. Our malicious exit node participated as a normal Tor router in the Tor network for the duration of the study (approximately six weeks). For our tests we did not actually make the exit server inject the JavaScript code; while this is a relatively trivial modification to the Tor code we used a simplified setup with a webserver serving pages with the JavaScript code already present.

The congestion part of the attack requires three components: a simple HTTP server serving an "infinite" stream of random data, a simple HTTP client downloading this stream of data via Tor, and finally a modified Tor client that constructs "long" circuits through those Tor nodes that the attacker would like to congest. Specifically, the modified Tor client allows the attacker to choose two (or more) routers with high bandwidth and a specific target Tor node, and build a long circuit by repeatedly alternating between the target node and the other high bandwidth nodes. The circuit is eventually terminated by connecting from some high-bandwidth exit node to the attacker's HTTP server which serves the "infinite" stream of random data as fast as the network can process it. As a result, the attacker maximizes the utilization of the Tor circuit. Naturally, an attacker with

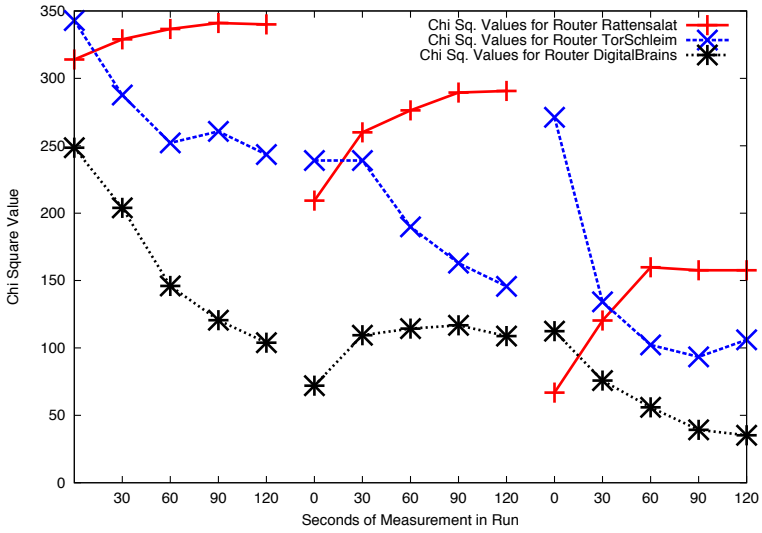

Figure 8: This graph shows three sets of cumulative χ^2 computations for three nodes; the actual entry node (`Rattensalat`), a node that initially shows up as a false-positive (`TorSchleim`) and a typical negative (`DigitalBrains`). As expected, the χ^2 values (at time 120 s) are consistently the highest for the correct node; false-positives can be ruled out through repeated measurements.

significant bandwidth can elect to build multiple circuits in parallel or build shorter circuits and still exhaust the bandwidth resources of the target Tor router.

In order to cause congestion, we simply started the malicious client Tor process with the three chosen Tor routers and route length as parameters and then attempted to connect via `libcurl` [6] to the respective malicious server process. The amount of data received was recorded in order to determine bandwidth consumed during the tests. In order to further increase the load on the Tor network the experiments presented actually used two identical attacker setups with a total of six machines duplicating the three machine setup described in the previous paragraph. We found path lengths of 24 (making our attack strength eight times the attacker bandwidth) sufficient to alter latencies. The overall strength of the attack was measured by the sum of the number of bytes routed through the Tor network by both attacker setups. For each trial, we waited to receive six hundred responses from the "victim"; since the browser transmitted requests to Tor at roughly one request per second, a trial typically took approximately ten minutes.

In addition to measuring the variance in packet arrival time while congesting a particular Tor router, each trial also included baseline measurements of the "uncongested" network to discover the normal variance in packet arrival time for a particular circuit. As discussed earlier, these baseline measurements are crucial for determining the significance of the effect that the congestion attack has had on the target circuit.

Fig. 5 illustrates how running the attack on the first hop of a circuit changes the latency of the received HTTP requests generated by the JavaScript code. The figure uses the same style chosen by Murdoch and Danezis [27], except that an additional line was added to indicate the strength of the attack (as measured by the amount of traffic provided by the adversary). For comparison, the first half of each of the figures shows the node latency variance when it is *not* under active congestion attack (or at least not by us).

While the plots in Fig. 5 visualize the impact of the congestion attack in a simple manner, histograms showing the variance in latency are more suitable to demonstrate the significance of the statistical difference in the traffic patterns. Fig. 6 shows the artificial delay experienced by requests traveling through the Tor network as observed by the adversary. Since Tor is a low-latency anonymization service, the requests group around a low value for a circuit that is not under attack. As expected, if the entry node is under attack, the delay distribution changes from a steep vertical peak to a mostly horizontal distribution. Fig. 6 also includes the best-fit linear approximation functions for the latency histograms which we use to characterize how vertical or how horizontal the histogram is as described in Section 3.2.

Fig. 7 illustrates how the χ^2 values evolve for various nodes over time. Here, we first characterized the baseline congestion for the router for five minutes. Then, the congestion attack was initiated (congesting the various guard nodes). For each attacked node, we used the modified

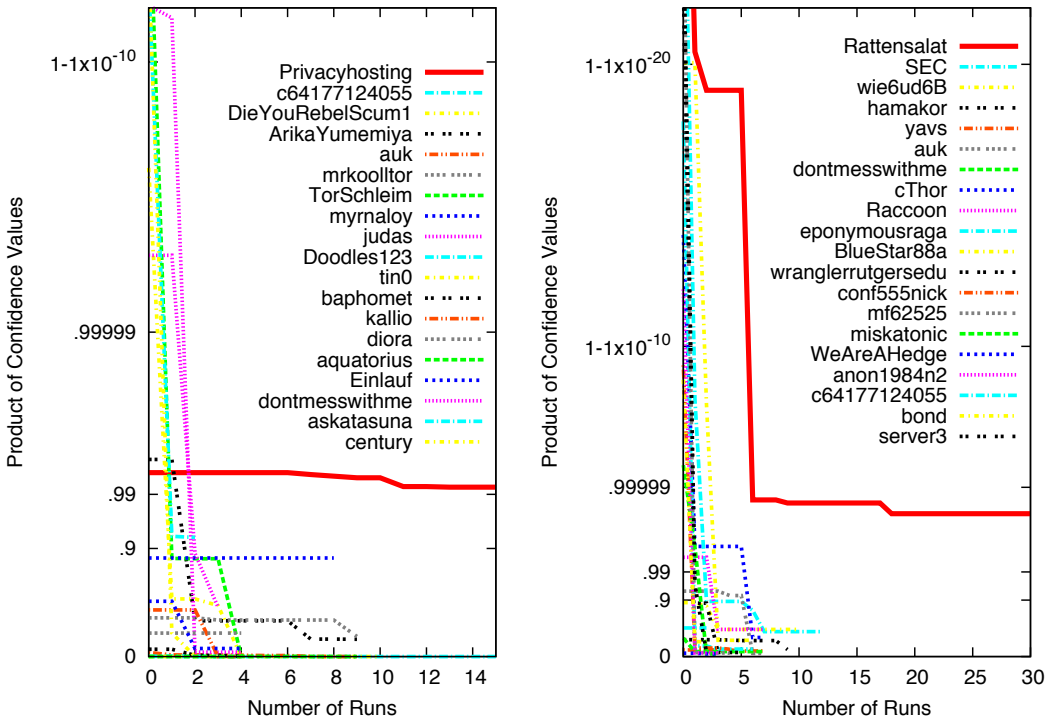

Figure 9: Plot of the product of χ^2 p-values for the top 20 candidate nodes (out of ∼200 and ∼250, respectively) by run (a run is 300 s baseline vs. 300 s attack) for two entry nodes. The first hop (`Privacyhosting` (left), `Rattensalat` (right)) routers were tested many more times than other routers, because the others quickly fell to low values. We expect an attacker to perform more measurements for routers that score high to validate the correct entry node was found. Our measurements demonstrate that the multiplied p-value remains consistently high for the correct entry node. The y-axis is plotted on a log scale from 0 to $1-1\times10^{-10}$ and $1-1\times10^{-20}$, respectively. We speculate that the lower maximum value for `Privacyhosting` is due to its higher bandwidth (900 kB/s vs. 231 kB/s).

χ^2 summation (from Section 3.3) to determine how congested the victim's circuit had become at that time. We computed (cumulative) χ^2 values after 30 s, 60 s, 90 s and so forth. For the χ^2 calculations, we used 60 bins for 300 baseline values; in other words, the time intervals for the bins were chosen so that each bin contained five data points during the five minutes of baseline measurement. The 20 bins in the middle were not included in the summation, resulting in 40 degrees of freedom. As expected, given only 30 s of attack data some "innocent" nodes have higher χ^2 values compared to the entry node (false-positives). However, given more samples the χ^2 values for those nodes typically drop sharply whereas the χ^2 value when congesting the entry node increases or remains high. Of course, false-positive nodes χ^2 values may increase due to network fluctuations over time as well.

Unlucky baseline measurements and shifts in the baseline latency of a router over time can be addressed by iterating between measuring baseline congestion and attack measurements. Fig. 8 shows three iterations of first determining the current baseline and then computing χ^2 values under attack. Again the correct entry node exhibits the largest χ^2 values each time after about a minute of gathering latency data under attack.

Given the possibility of false-positives showing up initially when computing χ^2 values, the attacker should target "all" suspected guard nodes for the first few iterations, and then focus his efforts on those nodes that scored highly. Fig. 9 illustrates this approach. It combines the data from multiple iterations of baseline measurements and χ^2 calculations from attack runs. The attacker determines for each χ^2 value the corresponding confidence interval. These values are frequently large (99.9999% or higher are not uncommon) since Tor routers do frequently experience significant changes in congestion. Given these individual confidence values for each individual iteration, a cumulative score is computed as the product[2] of these values. Fig. 9 shows the Tor

[2]It is conceivable that multiplying χ^2 values may cause false-

Router	Πp	r	Peak BW	Configured BW
Rattensalat	0.999991	44	231 kB/s	210 kB/s
c64177124055	0.903	3	569 kB/s	512 kB/s
Raccoon	0.891	8	3337 kB/s	4100 kB/s
wie6ud6B	0.890	11	120 kB/s	100 kB/s
SEC	0.870	13	4707 kB/s	5120 kB/s
cThor	0.789	8	553 kB/s	500 kB/s
BlueStar88a	0.734	7	111 kB/s	100 kB/s
bond	0.697	3	407 kB/s	384 kB/s
eponymousraga	0.458	7	118 kB/s	100 kB/s
conf555nick	0.450	5	275 kB/s	200 kB/s

Table 1: This table lists the top ten (out of 251 total) products of confidence intervals (p-values). r is the number of iterations (and hence the number of factors in Πp) that was performed for the respective router. As expected, the entry node Rattensalat achieves the highest score.

routers with the highest cumulative scores using this metric from trials on two different entry nodes. Note that fewer iterations were performed for routers with low cumulative scores; the router with the highest score (after roughly five iterations) and the most overall iterations is the correctly identified entry node of the circuit.

Table 1 contrasts the product of χ^2 values (as introduced in Section 3.3) obtained while attacking the actual first hop with the product while attacking other Tor routers. The data shows that our attack can be used to distinguish the first hop from other routers when controlling the exit router (therefore knowing a priori the middle router).

Finally, by comparing the highest latency observed during the baseline measurement with the highest latency observed under attack, Table 2 provides a simple illustration showing that the congestion attack actually has a significant effect.

5 Proposed Solutions

An immediate workaround that would address the presented attack would be disabling of JavaScript by the end user. However, JavaScript is not the only means by which an attacker could obtain timing information. For example, redirects embedded in the HTML header could also be used (they would, however, be more visible to the end user). Links to images, frames and other features of HTML could also conceivably be used to generate repeated requests. Disabling all of these features has the disadvantage that the end user's browsing experience would suffer.

negatives should a single near-zero χ^2 value for the correct entry node be observed. While we have not encountered this problem in practice, using the mean of χ^2 values would provide a way to avoid this theoretical problem.

A better solution would be to thwart the denial-of-service attack inherent in the Tor protocol. Attackers with limited bandwidth would then no longer be able to significantly impact Tor's performance. Without the ability to selectively increase the latency of a particular Tor router, the resulting timing measurements would most likely give too many false positives. We have extended the Tor protocol to limit the length of a path. The details are described in [9]; we will detail the key points here.

In the modified design, Tor routers now must keep track of how often each circuit has been extended and refuse to route messages that would extend the circuit beyond a given threshold t. This can be done by tagging messages that *may* extend the circuit with a special flag that is not part of the encrypted stream. The easiest way to do this is to introduce a new Tor cell type that is used to flag cells that may extend the circuit. Routers then count the number of messages with the special flag and refuse to route more than a given small number (at the moment, eight) of those messages. Routers that receive a circuit-extension request check that the circuit-extension message is contained in a cell of the appropriate type. Note that these additional checks do not change the performance characteristics of the Tor network. An attacker could still create a long circuit by looping back to an adversary-controlled node every t hops; however, the adversary would then have to provide bandwidth to route every t-th packet; as a result, the bandwidth consumption by the attacker is still bounded by the small constant t instead of the theoretically unbounded path length m.

While this change prevents an attacker from constructing a circuit of arbitrary length, it does not fully prevent the attacker from constructing a path of arbitrary length. The remaining problem is that the attacker could establish a circuit and then from the exit node reconnect to the Tor network again as a client. We could imagine config-

Router Attacked	Max Latency Difference	Avg. Latency Difference	Runs
Rattensalat	70352 ms	25822 ms	41
Wiia	46215 ms	470 ms	5
downtownzion	39522 ms	2625 ms	9
dontmesswithme	37648 ms	166 ms	8
wie6ud6B	35058 ms	9628 ms	9
TorSchleim	28630 ms	5765 ms	15
hamakor	25975 ms	6532 ms	8
Vault24	24330 ms	4647 ms	7
Einlauf	22626 ms	2017 ms	8
grsrlfz	22545 ms	10112 ms	2

Table 2: This table shows the top 10 highest latency differences between the maximum observed measurement in attack runs versus the baseline runs for each router. Unsurprisingly, the difference between the maximum latency observed during the congestion attack and the baseline measurement is significantly higher when attacking the correct first hop compared to attacking other routers. Also included for comparision is the average max latency over all iterations (also higher for the correct first hop), and the number of runs.

uring all Tor relays to refuse incoming connections from known exit relays, but even this approach does not entirely solve the problem: the attacker can use any external proxies he likes (e.g. open proxies, unlisted Tor relays, other anonymity networks) to "glue" his circuits together. Assuming external proxies with sufficient aggregate bandwidth are available for gluing, he can build a chain of circuits with arbitrary length. Note that the solution proposed in [30] — limiting circuit construction to trees — does not address this issue; furthermore, it increases overheads and implementation complexity far beyond the change proposed here and (contrary to the claims in [30]) may also have an impact on anonymity, since it requires Tor to fundamentally change the way circuits are constructed. We leave a full solution to this problem as an open research question.

Finally, given that strong adversaries may be able to mount latency altering attacks without Tor's "help", Tor users might consider using a longer path length than the minimalistic default of three. This would involve changes to Tor, as currently the only way for a user to change the default path length would be to edit and recompile the code (probably out of scope for a "normal" user). While the presented attack can be made to work for longer paths, the number of false positives and the time required for a successful path discovery increase significantly with each extra hop. Using a random path length between four and six would furthermore require the adversary to confirm that the first hop was actually found (by determining that none of the other Tor routers could be a predecessor). Naturally, increasing the path length from three to six would significantly increase the latency and bandwidth requirements of the Tor network and might also hurt with respect to other attacks [2].

6 Low-cost Traffic Analysis Failure Against Modern Tor

We attempted to reproduce Murdoch and Danezis's work [27] on the Tor network of 2008. Murdoch provided us with their code and statistical analysis framework which performs their congestion attack while measuring the latency of the circuit. Their analysis also determines the average latency and uses normalized latencies as the strength of the signal.

The main difference in terms of how data is obtained between Murdoch and Danezis and the attack presented in Section 3 is that Murdoch and Danezis use a circuit constructed by the attacker to measure the latency introduced by the victim circuit whereas our attack uses a circuit constructed by the victim to measure the latency introduced by the attacker.

As in this paper, the adversary implemented by Murdoch and Danezis repeatedly switches the congestion attack on and off; a high correlation between the presence of high latency values and the congestion attack being active is used to determine that a particular router is on the circuit. If such a correlation is absent for the correct router, the attack produces false negatives and fails. If a strong correlation is present between high latency values and random time periods (without an active attack) then the attack produces false positives and also fails.

Fig. 10 shows examples of our attempts at the method used in [27], two with the congestion attack being active and two without. Our experiments reproduced Murdoch and Danezis's attack setup where the attacker tries to measure the congestion caused by the victim's circuit. Note that in the graphs on the right, the congestion attack was run against a Tor router unrelated to the circuit

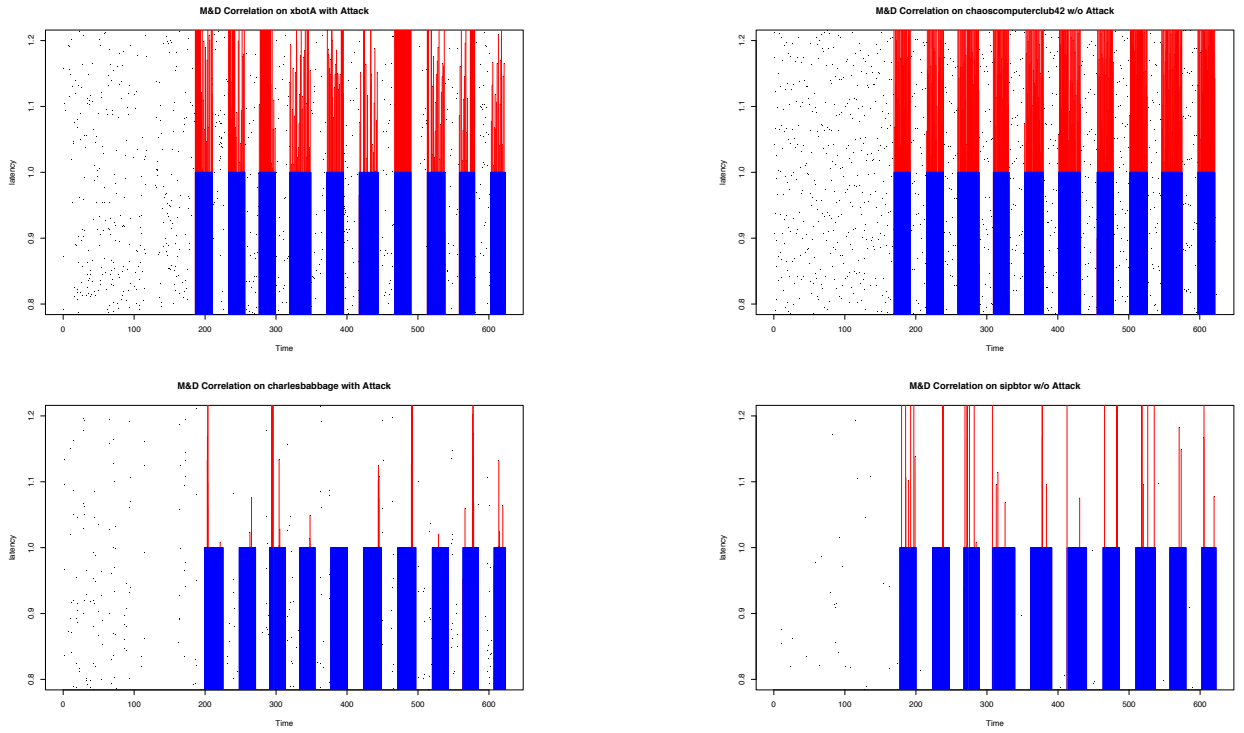

Figure 10: These graphs show four runs of the method used in [27], two with the congestion attack being active (on the left) and two without (on the right). The figure plots the observed latency of a router over time. Blue bars are used to indicate when the congestion attack was active; in the case of the graphs on the right the attack was active on an unrelated circuit. Red lines are drawn to latency values above average to mark latencies that correlate with the attack, according to the Murdoch and Danezis style analysis.

and thus inactive for the circuit that was measured. Any correlation observed in this case implies that Murdoch and Danezis's attack produces false positives. The "visual" look of the graphs is the same whether the attack is targeted at that relay or not. Specifically, the graphs on the right suggest a similar correlation pattern even when the attack was "off" (or targeting unrelated Tor routers). This is due to the high volume of traffic on today's Tor network causing baseline congestion which makes their analysis too indiscriminate.

Table 3 shows some representative correlation values that were computed using the statistical analysis from [27] when performed on the modern Tor network. Note that the correlation values are high regardless of whether or not the congestion attack was actually performed on the respective router. For Murdoch and Danezis's analysis to work, high correlation values should only appear for the attacked router.

The problem with Murdoch and Danezis's attack and analysis is not primarily with the statistical method; the single-circuit attack itself is simply not generating a sufficiently strong signal on the modern network. Fig. 11 plots the baseline latencies of Tor routers as well as the latencies of routers subjected to Murdoch and Danezis's congestion attack in the style we used in Fig. 6. There are hardly any noticeable differences between routers under Murdoch and Danezis's congestion attack and the baseline. Fig. 12 shows the latency histograms for the same data; in contrast to the histograms in Fig. 6 there is little difference between the histograms for the baseline and the attack.

In conclusion, due to the large amount of traffic on the modern Tor network, Murdoch and Danezis's analysis is unable to differentiate between normal congestion and congestion caused by the attacker; the small amount of congestion caused by Murdoch and Danezis is lost in the noise of the network. As a result, their analysis produces many false positives and false negatives. While these experiments only represent a limited case-study and while Murdoch and Danezis's analysis may still work in some cases, we never got reasonable results on the modern Tor network.

Router	Correlation	Attacked?	Peak BW	Configured BW
morphiumpherrex	1.43	Yes	222 kB/s	201 kB/s
chaoscomputerclub23	1.34	No	5414 kB/s	5120 kB/s
humanistischeunion1	1.18	No	5195 kB/s	6000 kB/s
mikezhangwithtor	1.07	No	1848 kB/s	2000 kB/s
hummingbird	1.03	No	710 kB/s	600 kB/s
chaoscomputerclub42	1.00	Yes	1704 kB/s	5120 kB/s
degaussYourself	1.00	No	4013 kB/s	4096 kB/s
ephemera	0.91	Yes	445 kB/s	150 kB/s
fissefjaes	0.99	Yes	382 kB/s	50 kB/s
zymurgy	0.86	Yes	230 kB/s	100 kB/s
charlesbabbage	0.53	Yes	2604 kB/s	1300 kB/s

Table 3: This table shows the correlation values calculated using the Murdoch and Danezis's attack on the Tor network in Spring of 2008. False positives and false negatives are both abundant; many naturally congested routers show a strong correlation suggesting they are part of the circuit when they are not.

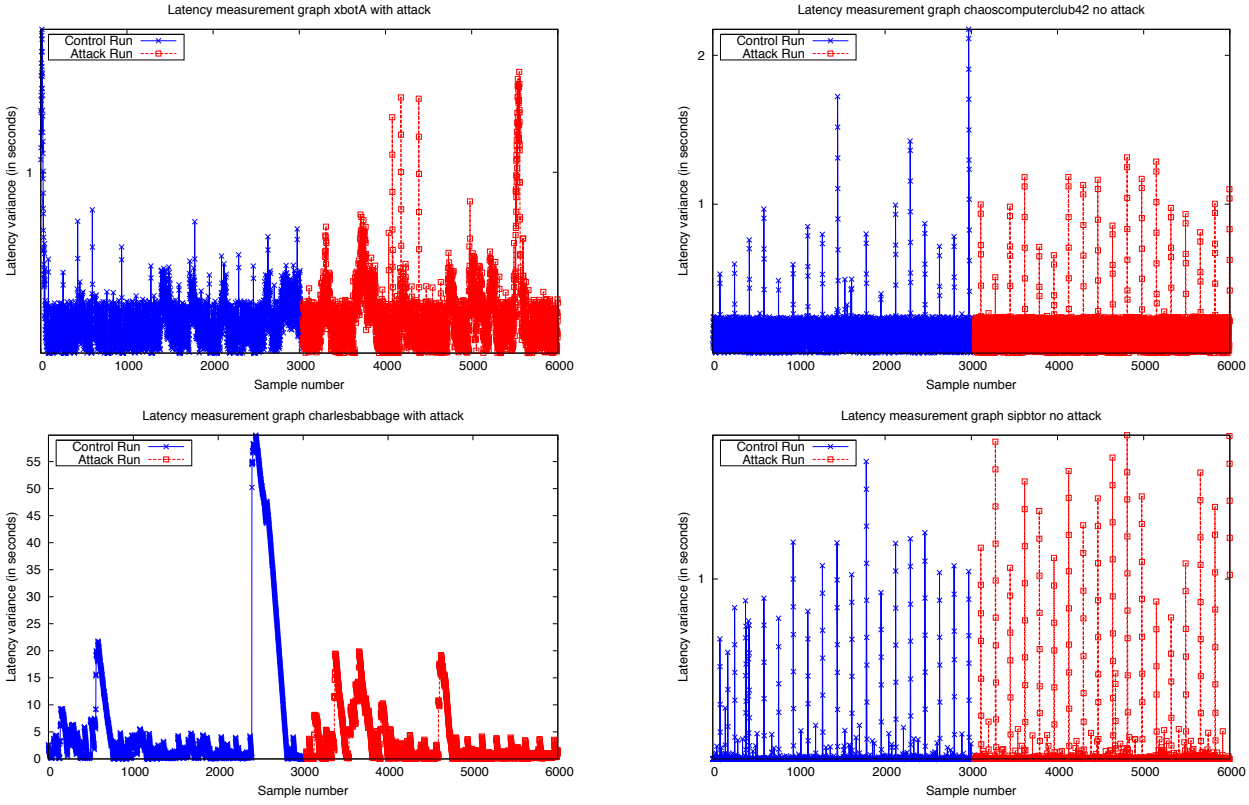

Figure 11: These graphs correspond to Fig. 10, showing the same attack in the style we used in Fig. 5. Note that during the attack phase the congestion circuit is turned on and off just as illustrated in Fig. 10. For all four routers the latency measurements are almost identical whether the attack was present or not.

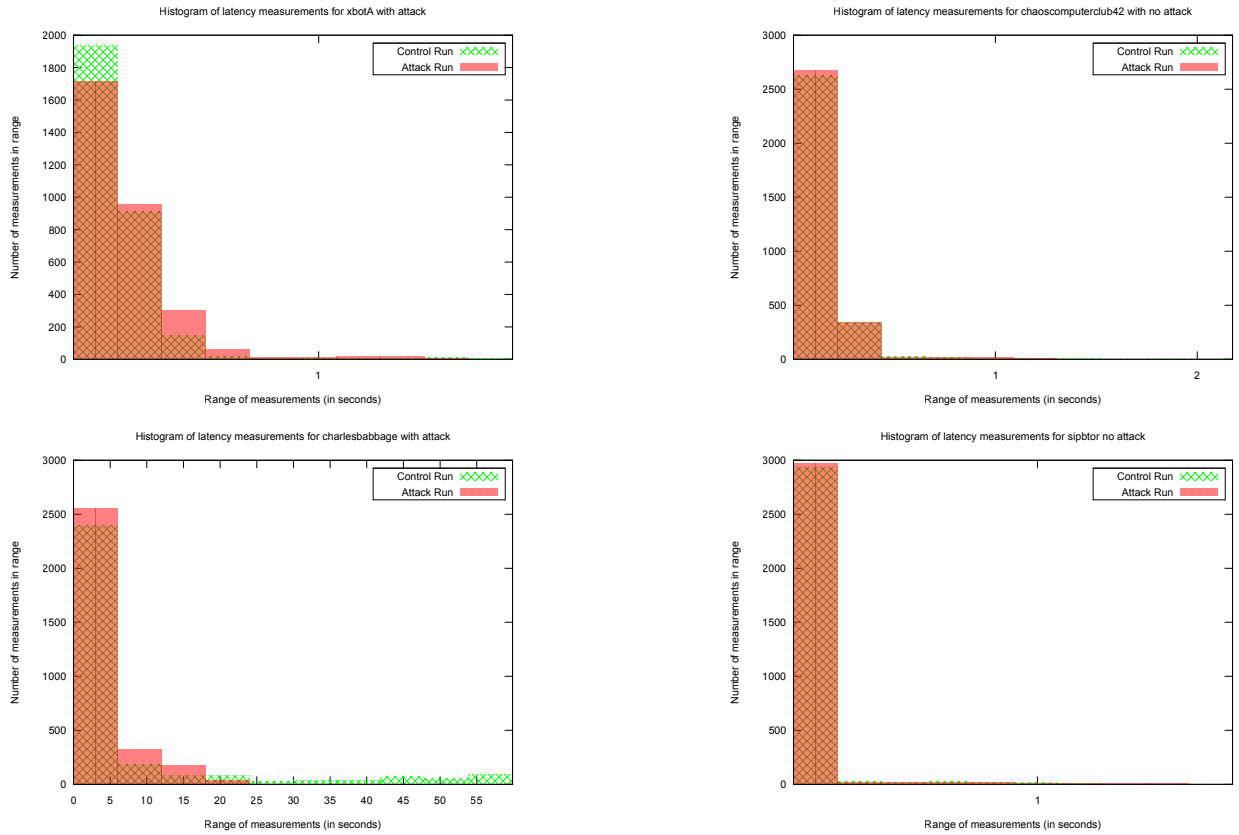

Figure 12: Once more we show the same data for comparison as shown in Fig. 10, this time in the histogram style we use in Fig. 6. The overlap between the control run and the attack run is difficult to see due to the similarity of latency distributions.

7 Conclusion

The possibility of constructing circuits of arbitrary length was previously seen as a minor problem that could lead to a DoS attack on Tor. This work shows that the problem is more serious, in that an adversary could use such circuits to improve methods for determining the path that packets take through the Tor network. Furthermore, Tor's minimalistic default choice to use circuits of length three is questionable, given that an adversary controlling an exit node would only need to recover a tiny amount of information to learn the entire circuit. We have made some minimal changes to the Tor protocol that make it more difficult (but not impossible) for an adversary to construct long circuits.

Acknowledgments

This research was supported in part by the NLnet Foundation from the Netherlands (`http://nlnet.nl/`) and under NSF Grant No. 0416969. The authors thank P. Eckersley for finding a problem in an earlier draft of the paper and K. Grothoff for editing.

References

[1] BACK, A., MÖLLER, U., AND STIGLIC, A. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proceedings of Information Hiding Workshop (IH 2001)* (April 2001), I. S. Moskowitz, Ed., Springer-Verlag, LNCS 2137, pp. 245–257.

[2] BORISOV, N., DANEZIS, G., MITTAL, P., AND TABRIZ, P. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security* (New York, NY, USA, October 2007), ACM, pp. 92–102.

[3] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM 24*, 2 (February 1981), 84–90.

[4] DAI, W. Two attacks against freedom. http://www.weidai.com/freedom-attacks.txt, 2000.

[5] DANEZIS, G., DINGLEDINE, R., AND MATHEWSON, N. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (May 2003), pp. 2–15.

[6] DANIEL STENBERG, E. A. libcurl, 1998–2009. Open Source C-based multi-platform file transfer library.

[7] DESMEDT, Y., AND KUROSAWA, K. How to break a practical MIX and design a new one. In *Advances in Cryptology — Eurocrypt 2000, Proceedings* (2000), Springer-Verlag, LNCS 1807, pp. 557–572.

[8] DIAZ, C., AND SERJANTOV, A. Generalising mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)* (March 2003), R. Dingledine, Ed., Springer-Verlag, LNCS 2760, pp. 18–31.

[9] DINGLEDINE, R. Tor proposal 110: Avoiding infinite length circuits. https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/110-avoid-infinite-circuits.txt, March 2007.

[10] DINGLEDINE, R. Tor bridges specification. Tech. rep., The Tor Project, https://svn.torproject.org/svn/tor/trunk/doc/spec/bridges-spec.txt, 2008.

[11] DINGLEDINE, R., AND MATHEWSON, N. Design of a blocking-resistant anonymity system. Tech. rep., The Tor Project, https://svn.torproject.org/svn/tor/trunk/doc/design-paper/blocking.pdf, 2007.

[12] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium* (August 2004).

[13] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. *RFC 2616: Hypertext Transfer Protocol — HTTP/1.1*. The Internet Society, June 1999.

[14] FREEDMAN, M. J., AND MORRIS, R. Tarzan: a peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security* (New York, NY, USA, November 2002), ACM, pp. 193–206.

[15] FREEDMAN, M. J., SIT, E., CATES, J., AND MORRIS, R. Introducing tarzan, a peer-to-peer anonymizing network layer. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems* (London, UK, 2002), Springer-Verlag, pp. 121–129.

[16] GOLDSCHLAG, D. M., REED, M. G., AND SYVERSON, P. F. Hiding Routing Information. In *Proceedings of Information Hiding: First International Workshop* (May 1996), R. Anderson, Ed., Springer-Verlag, LNCS 1174, pp. 137–150.

[17] GÜLCÜ, C., AND TSUDIK, G. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96* (February 1996), IEEE, pp. 2–16.

[18] HAN, J., AND LIU, Y. Rumor riding: Anonymizing unstructured peer-to-peer systems. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols* (Washington, DC, USA, Nov 2006), IEEE Computer Society, pp. 22–31.

[19] HOPPER, N., VASSERMAN, E. Y., AND CHAN-TIN, E. How much anonymity does network latency leak? In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security* (New York, NY, USA, October 2007), ACM, pp. 82–91.

[20] KEIL, F., SCHMIDT, D., ET AL. Privoxy - a privacy enhancing web proxy. http://www.privoxy.org/.

[21] KESDOGAN, D., EGNER, J., AND BÜSCHKES, R. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of the Second International Workshop on Information Hiding* (London, UK, 1998), Springer-Verlag, LNCS 1525, pp. 83–98.

[22] LANDSIEDEL, O., PIMENIDIS, A., WEHRLE, K., NIEDERMAYER, H., AND CARLE, G. Dynamic multipath onion routing in anonymous peer-to-peer overlay networks. *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE* (Nov. 2007), 64–69.

[23] LEVINE, B. N., REITER, M. K., WANG, C., AND WRIGHT, M. K. Timing attacks in low-latency mix-based systems. In *Proceedings of Financial Cryptography (FC '04)* (February 2004), A. Juels, Ed., Springer-Verlag, LNCS 3110, pp. 251–265.

[24] MCLACHLAN, J., AND HOPPER, N. Don't clog the queue! circuit clogging and mitigation in p2p anonymity schemes. In *Financial Cryptography* (2008), G. Tsudik, Ed., vol. 5143 of *Lecture Notes in Computer Science*, Springer, pp. 31–46.

[25] MÖLLER, U., COTTRELL, L., PALFRADER, P., AND SASSAMAN, L. Mixmaster Protocol — Version 2. IETF Internet Draft, December 2004.

[26] MURDOCH, S. J. *Covert channel vulnerabilities in anonymity systems*. PhD thesis, University of Cambridge, December 2007.

[27] MURDOCH, S. J., AND DANEZIS, G. Low-cost traffic analysis of Tor. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy* (Washington, DC, USA, May 2005), IEEE Computer Society, pp. 183–195.

[28] NAMBIAR, A., AND WRIGHT, M. Salsa: a structured approach to large-scale anonymity. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security* (New York, NY, USA, October 2006), ACM, pp. 17–26.

[29] ØVERLIER, L., AND SYVERSON, P. Locating hidden servers. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy* (Washington, DC, USA, May 2006), IEEE Computer Society, pp. 100–114.

[30] PAPPAS, V., ATHANASOPOULOS, E., IOANNIDIS, S., AND MARKATOS, E. P. Compromising anonymity using packet spinning. In *Proceedings of the 11th Information Security Conference (ISC 2008)* (2008), T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, Eds., vol. 5222 of *Lecture Notes in Computer Science*, Springer, pp. 161–174.

[31] PERRY, M., AND SQUIRES, S. https://www.torproject.org/torbutton/, 2009.

[32] PFITZMANN, A., PFITZMANN, B., AND WAIDNER, M. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems* (February 1991), pp. 451–463.

[33] RENNHARD, M., AND PLATTNER, B. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society* (New York, NY, USA, November 2002), ACM, pp. 91–102.

[34] SERJANTOV, A., DINGLEDINE, R., AND SYVERSON, P. From a trickle to a flood: Active attacks on several mix types. In *IH '02: Revised Papers from the 5th International Workshop on Information Hiding* (London, UK, 2003), F. Petitcolas, Ed., Springer-Verlag, LNCS 2578, pp. 36–52.

[35] SHMATIKOV, V., AND WANG, M.-H. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of the 11th European Symposium on Research in Computer Security (ESORICS)* (September 2006), pp. 236–252.

[36] WIANGSRIPANAWAN, R., SUSILO, W., AND SAFAVI-NAINI, R. Design principles for low latency anonymous network systems secure against timing attacks. In *Proceedings of the fifth Australasian symposium on ACSW frontiers (ACSW '07)* (Darlinghurst, Australia, Australia, 2007), Australian Computer Society, Inc, pp. 183–191.