Predictive Black-Box Mitigation of Timing Channels

Aslan Askarov
aslan@cs.cornell.edu

Danfeng Zhang
zhangdf@cs.cornell.edu

Andrew C. Myers
andru@cs.cornell.edu

Department of Computer Science
Cornell University
Ithaca, NY 14853

ABSTRACT

We investigate techniques for general black-box mitigation of timing channels. The source of events is wrapped by a timing mitigator that delays output events so that they contain only a bounded amount of information. We introduce a general class of timing mitigators that can achieve any given bound on timing channel leakage, with a tradeoff in system performance. We show these mitigators compose well with other mechanisms for information flow control, and demonstrate they are effective against some known timing attacks.

Categories and Subject Descriptors: C.2.0 [Computer Communication Networks]: General—*Security and protection*

General Terms: Security

Keywords: Timing channels, mitigation, information flow

1. INTRODUCTION

Controlling timing channels is difficult but important. The difficulty has long been recognized [20, 9, 27], but their importance has been reinforced by recent work that shows timing channels can quickly leak sensitive information. Attacks exploit the timing of cryptographic operations [17, 4], of cache operations [26], and of web server responses [2]. These attacks work even without cooperation of any software on the system being timed. If the system contains malicious code or hardware (e.g., [30]), timing can also be exploited as a robust covert channel [21].

In complex computing systems, different computations affect each others' timing through shared resources such as caches, the processor, the disk, and the network. The precise time of an event may depend on many pieces of sensitive information and computation. Therefore, timing measurements act as a kind of antenna receiving signals from throughout the system. The combined signal might be analyzed by a sufficiently clever adversary to learn about any of the information influencing timing.

A useful distinction to draw is between *internal* and *external* timing channels. Internal timing channels occur when time is explicitly or implicitly measured from within the system that contains timing channels. External timing channels are measured from outside the system, and are therefore less easy to control. A variety of methods have been proposed for mitigating or preventing internal timing channels (e.g., [14, 1, 33]); this paper focuses on external timing channels, where the adversary has more power: the ability to accurately time externally visible behavior of the system. Prior techniques for preventing timing channels fall into two camps. One approach that has not proved effective in general is to add random delays to timing-sensitive operations [11] or to timing measurements [14]. Randomness adds noise to the timing signal, reducing but not eliminating the bandwidth of the timing channel. Further, stealthy timing channels robust to noise can be constructed [21].

A better, non-random method is to pad the run time of sensitive operations to a fixed time, or to a multiple of a fixed time. Padding mitigates timing leaks but does not eliminate them entirely if the padded operation can take more than the allotted quantum of time. For cryptographic operations, a correlation between sensitive input and the run time can also be addressed by *blinding* techniques [5, 17] that unpredictably change the input, but blinding is not applicable to general computations.

This paper introduces a more general scheme for mitigating timing channels. Unlike cryptographic blinding, this scheme applies to a broad class of computing systems and computations. It does not prevent timing leaks, which seems to be impossible in the general case. Rather, it bounds the amount of information leaked through the timing channel as a function of elapsed time. We show that simple mitigation schemes can ensure that no more than $\log^2(t)$ bits of information are leaked, where t is the running time (all logarithms in this paper are base 2). Further, an arbitrary bound on information leakage can be enforced, down to as low as $O(\log(t))$. However, tighter bounds have a price: they can reduce system throughput and increase system latency, particularly if the system has unpredictable behavior.

This paper makes the following contributions:

1. It introduces a novel, principled way to mitigate external timing channels.
2. This mitigation scheme is shown to enforce a specified bound on the amount of information leaked.
3. The new mitigation scheme is demonstrated to defeat some timing attacks discovered in prior work, and to provide good performance in some cases.
4. Total leakage is shown to be bounded when timing mitigation is combined with other information flow control mechanisms.

The rest of the paper is structured as follows. Section 2 introduces a simple version of the mitigation scheme. This simple

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.

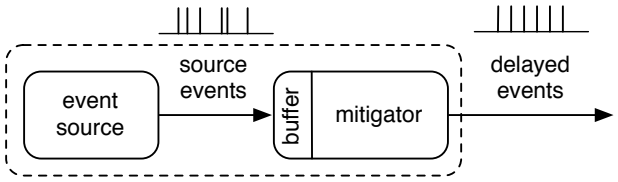


Figure 1: System overview

scheme is an instance of a more general framework for provable mitigation, defined in Section 3. Section 4 empirically explores how mitigation scheme affects the performance of the underlying system. The scheme is shown in Section 5 to defend against some known timing attacks. Related work is discussed in Section 6, and Section 7 concludes.

2. SOME SIMPLE MITIGATION SCHEMES

2.1 System model

We begin with a simple model of a computing system that produces externally observable events whose timing may be a channel. Because the mitigation scheme works regardless of the internal details of computation, the computing system is treated simply as a black-box source of events. As depicted in Figure 1, the event source generates events that are delayed by the mitigation mechanism so that their times of delivery convey less information. Delaying events while preserving their order is the only behavior of the mitigator that we consider.

We ignore for now the attributes of events other than time. These attributes include the actual content of an event and also the choice of communication medium (e.g., different networks, or even visual displays or sound) over which it can be conveyed. Both content and choice of medium can be viewed as storage channels [20], which we assume are controlled by other means. Therefore we assume that the only information requiring control is encoded in the times at which events arrive from the source. This separate treatment of timing and storage channels is justified in Section 3.5.

We assume the attacker observes delayed events and knows the design of the mitigator though not its internal state. The goal of the attacker is to communicate information from inside the event source to the outside. Therefore the attacker consists of two parts: an *insider* that controls the timing of source events, and an external *observer* that attempts to learn sensitive information from this timing channel. The *content* of the events may also be observable to the attacker, which motivates our choice to not have the mitigator generate dummy events. The observer may combine information from both the content and timing of messages. In real world this corresponds to attacker-controlled software that communicates seemingly benign messages on a storage channel, but transmits sensitive information using timing.

As shown in the figure, it is useful to allow the mitigation system to buffer events in a queue so the event source can run ahead, generating more events without waiting. We consider adding input events to the system model in Section 3.6.

2.2 Quantizing time

A very simple mitigation scheme that has been explored in prior work [13, 11, 4] permits events to leave the mitigator only at scheduled times that are multiples of a particular time quantum q. We refer to the times when events are permitted as *slots*, which in this case occur at times $q, 2q$, etc. Without loss of generality, let us use $q = 1$ to analyze this scheme.

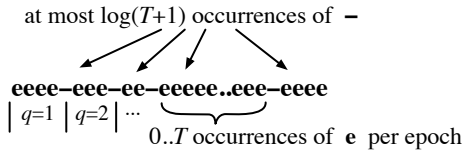
Suppose we allow the system to run for time T, and during that time there is an event ready to be delivered in every slot except that at some point the event source may stop producing events (effectively, it terminates). The total number of events delivered must be an integer between 0 and T. Because all the slots filled with events precede all the empty slots, the external observer can make at most $T+1$ possible distinct observations. According to information theory, the maximum amount of information that can be transmitted by one of $T + 1$ possible observations is achieved when the possible observations are uniformly distributed. This value, in bits, is the log base 2 of the number of possible observations, or $\log(T + 1)$. For $q \neq 1$, it is $\log(\frac{T+1}{q})$.

2.3 A basic mitigation scheme

An asymptotically logarithmic bound on leakage sounds appealing, but in general we cannot count on the event source to fill every slot with an event. In the general case, maximum leakage from the simple quantizing approach is one bit per quantum, leading to an unpleasant tradeoff between security and performance.

However, a sublinear (in fact, polylogarithmic) bound is achievable even if the event source misses some slots. Perhaps the simplest way to achieve this is to double the quantum q every time a slot is missed. Doubling the quantum ensures that in time T there can be at most $\log(T + 1)$ misses. Effectively, the event source is penalized for irregular behavior. For the penalty will be effective, the multiplicative factor need not be 2; the number of misses will grow logarithmically for any multiplicative factor greater than 1.

We can represent all behaviors of the resulting system as strings constructed from the symbols **e** (for an event that fills a slot) and $-$ (for a missed slot). A given string generated by the regular expression $(\mathbf{e}|-)^*$ precisely determines the times at which events emerge from the mitigator, so the distinct strings correspond exactly to the possible external timing observations. Therefore, the maximum of the expected number of bits of information transmitted by time T is the log base 2 of the number of strings that can be observed within time T. These strings contain at most $\log(T+1)$ occurrences of $-$. Between and around these occurrences are consecutive sequences of between 0 and T filled slots (**e**'s), as suggested by this figure:



Each sequence of **e**'s falls into a different *epoch* with its own characteristic quantum. There are at most $\log(T + 1) + 1$ epochs, so the number of possible strings observable within time T is at most $(T+1)^{\log(T+1)+1}$. The maximum information content of the timing channel is the log of this number, or $\log(T + 1) \cdot (\log(T + 1) + 1) = \log^2(T + 1) + \log(T + 1)$. This is bounded above by $(1+\epsilon)\log^2 T$ where ϵ can be made arbitrarily small for sufficiently large T. With the more careful combinatorial analysis given in the appendix, we can show leakage is bounded by $O(\log T(\log T - \log\log T))$. In either case, timing leakage is $O(\log^2(T))$, which is a slowly growing function of time.

2.4 Slow-doubling mitigation

Doubling on every miss performs poorly if the event source is quiescent for long periods. The quantum-doubling scheme can be refined further to accommodate quiescent periods, by doubling the quantum only when a missed slot follows a filled slot (that is, a $-$ after an **e**). With this mitigator, no performance penalty is suffered

by an event source that is initially quiescent, but then generates all its output in a rapid series of events.

In this case we have epochs consisting of sequences like "−−−−" and "**eeee**". There can be at most $2\log(T+1)$ epochs, and there can be at most T strings per epoch, so the information content of the channel is no more than $2\log(T)\log(T+1) \leq (2+\epsilon)\log^2 T$. Thus, slow doubling gives much more flexibility without changing asymptotic information leakage.

In the next section we see that both the fast and slow doubling schemes are instances of a more general framework for epoch-based timing mitigation, enabling further important refinements such as adaptively reducing the quantum.

3. GENERAL EPOCH-BASED MITIGATION

The common feature of the mitigation schemes introduced thus far is that the mitigator divides time into epochs. During each epoch the mitigator operates according to a fixed schedule that predicts the future behavior of the event source. As long as the schedule predicts behavior accurately, the event source leaks no timing information except for the length of the epoch. However, a misprediction by the mitigator causes it to construct a new schedule; because this choice is in general observable by the adversary, some information leaks.

We can describe the mitigation schemes seen so far in these terms. For example, the slow doubling scheme has "**e**" epochs in which the mitigator predicts there will be an event ready for slots spaced at the current quantum q. It also has "−" epochs in which the mitigator predicts there will be no event ready for slots spaced at the quantum q. On a mispredicted slot (a miss) during an "**e**" epoch, the mitigator switches to a "−" epoch with a doubled quantum.

Let us now explore this framework more formally, to enable generating and analyzing a variety of mitigation schemes that meet specified bounds on timing channel transmission.

3.1 Mitigation

The mitigator is oblivious to the content of the events and does not alter their content. From the mitigator's point of view, source events and delayed events are considered as timestamps at which the events are received and delivered respectively.

Let source events be denoted by a monotonic sequence $s_1 \ldots s_n$, where $0 \leq s_1$ and each s_i specifies when the i-th event is received by the mitigator. We denote the mitigator by M. Given a sequence of source events $s_1 \ldots s_n$, let $\mathsf{M}(s_1 \ldots s_n)$ be the sequence of possibly delayed timestamps $d_1 \ldots d_m$ produced by the mitigator. The sequence is again monotonically increasing; also, we have $m \leq n$ and $s_i \leq d_i$. The last inequality means the mitigator cannot produce events before they are received from the source.

A mitigation scheme is *online* if the delayed sequence does not depend on timing or contents of future source messages. In this work we are only interested in online mitigation schemes.

Timing leakage.

Because timing of the events may depend on the sensitive data at the source, any variation in observed event timing creates an information channel. The larger the number of different observable variations, the more information can be transmitted over this channel. When events are mitigated, the number of possible sequences of events that a mitigator M can deliver by time T is

$$\mathbf{M}(T) = |\{d_1 \ldots d_m = \mathsf{M}(s_1 \ldots s_n) \mid d_m \leq T\}|$$

The amount of information that can be leaked by such mitigator, when the running time is bounded by T, is a logarithm of $\mathbf{M}(T)$.

DEFINITION 1 (LEAKAGE OF THE MITIGATOR). *Given a mitigator* M*, let leakage of* M *be* $\log \mathbf{M}(T)$.

This definition implicitly assumes that the mitigator can control the timing of events with perfect precision, but also credits the adversarial observer with perfect measurement abilities. More realistically, we can assume that the mitigator can control timing to at least the measurement precision of the observer, in which case the above formula still bounds leakage.

Bounding leakage.

We specify the security requirements for timing leakage as a bound, expressed as a function on running time T.

DEFINITION 2 (BOUNDING MITIGATOR LEAKAGE). *Given a mitigator* M*, and a leakage bound* $B(T)$*, we say that the leakage of* M *is bounded by* $B(T)$ *if for all* T*, we have* $\log \mathbf{M}(T) \leq B(T)$.

3.2 Epoch-based mitigation

In this work we focus on a specific class of mitigators that we dub *epoch-based* mitigators. An *epoch* represents a period of time during which the behavior of the mitigator meets the epoch *schedule*.

An epoch schedule is a sequence of epoch *predictions*, one for each slot. Epoch predictions can be either *positive* or *negative*. A positive prediction, denoted by $[t]^+$, means the mitigator expects to be able to deliver an event at time t. A negative prediction, denoted by $[t]^-$, says that no source events are expected to be available for delivery at time t. We may simply write t when the sign of the prediction is not important for the context. A prediction is an element of $\mathbb{R} \times \{+, -\}$, because times t are real-valued. An epoch schedule S is therefore a function from slot indices (from the natural numbers $\mathbb{N}$) to predictions.

DEFINITION 3 (EPOCH SCHEDULE). *An epoch schedule is a function* $S : \mathbb{N} \to \mathbb{R} \times \{+, -\}$*, where* $S(n)$ *is a* prediction *for the* n*-th slot in the epoch.*

We say that a positive prediction $S(n) = [t]^+$ *holds* or *is valid* if at time t the mitigator can deliver an event; in this case, this is also the n-th event in the epoch. A negative prediction $S(n) = [t]^-$ holds when no source events are available at time t.

Conversely, *failing* a positive prediction $[t]^+$ means that there are no events (available or buffered) to be delivered at time t. Failing a negative prediction $[t]^-$ means that there are buffered source events that have not yet been delivered by time t.

When a mitigator prediction $S(n)$ fails at the n-th slot, we observe an *epoch transition*. In addition to prediction failure, an epoch transition may be caused by *mitigator adjustments*. For example, the mitigator might adjust for a faster rate of source events, or might improve performance by flushing or partially flushing the buffer queue. We can now formally define an epoch:

DEFINITION 4 (EPOCH). *An epoch is a triple* (τ, τ', S) *where timestamps* τ *and* τ' *correspond to the beginning and the end of the epoch, and* S *is the epoch schedule.*

When the number of the epoch is important we write S_N for the schedule in epoch N.

Example.

Revisiting the basic doubling scheme from Section 2.3, we see that the prediction for every N-th epoch that starts at time t is given by the function $S_N(i) = [t + i \cdot 2^N]^+$.

For the slow doubling scheme of Section 2.4, every odd prediction is positive—it expects the events to be delivered at regular intervals, and every even prediction is negative—no events are expected from the source. These predictions can be expressed as follows:

$$S_N(i) = \begin{cases} [t + i \cdot 2^k]^+ & \text{if } N = 2k - 1 \\ [t + i \cdot 2^k]^- & \text{if } N = 2k \end{cases}$$

On the form of schedules.

Most of the examples of schedules in this paper are *constant-quantum* functions, where prediction times depend linearly on the epoch sequence number of the events. However, when timing pattern of the source events is well-understood, a finer prediction, described by an arbitrary function, could yield better performance. From the standpoint of security, the form of the schedule is irrelevant as long as the mitigator satisfies the leakage bound discussed in Section 3.4.

3.3 Leakage of epoch-based mitigators

Epoch-based design allows us to reduce the analysis of epoch-based mitigation to the analysis of individual epochs and of the transitions between them.

Variations within an epoch.

Because prediction times during an epoch are deterministic, the only source of timing variation within an epoch is the number of valid predictions. The latter is the key element in bounding the number of possible event sequences within an epoch. The number of valid predictions is bounded by the duration of the epoch, which itself is bounded by the current running time $T + 1$. Therefore the current running time $T + 1$ is a bound on the number of variations within each epoch.

Transition variations.

Epoch transitions may depend on source events too. Therefore, one needs to take into account the number of possible schedules for the next epoch. We denote by Λ_N the number of possible schedules when transitioning from epoch N to epoch $N + 1$.

The exact number of transition variations depends on the particular mitigation scheme. In the two schemes described thus far, the transition into a new epoch occurs only when a miss occurs, and only one new schedule is possible; hence, for all epochs N, we have $\Lambda_N = 1$.

An example mitigator for which Λ_N is greater than 1 is an *adaptive scheme* that uses the average rate of the previously received source events to choose the new schedule. In this case, Λ_N can be bounded by the current running time $T + 1$.

Section 4.1 describes the convergence experiment where the number of possible predictions for a given epoch is chosen from a fixed table and is exactly 2.

Bound on the number of total variations.

Consider an epoch-based mitigator at time T that has reached at most N epochs. Assume that within each epoch the number of variations is at most $T + 1$, and the number of possible transition variations into epoch i is Λ_i, where i ranges from 1 to N. We include Λ_N to accommodate the transition from epoch N to epoch $N + 1$ at time T. We can bound the number of possible variations of such a mitigator by a function $\mathbf{M}(T, N)$:

$$\mathbf{M}(T, N) = (T + 1)^N \cdot \Lambda_1 \cdot \Lambda_2 \cdot \ldots \cdot \Lambda_N$$

The leakage of this mitigator is bounded by the logarithm

$$\log \mathbf{M}(T, N) = N \log(T + 1) + \sum_{j=1}^{N} \log \Lambda_j \tag{1}$$

We refer to the term $\log(T + 1)$ as *epoch leakage* and to the terms $\log \Lambda_i$ as *transition leakage*.

Basic schemes revisited.

Revisiting the simple mitigators from Section 2, we see that because $\Lambda_N = 1$, the leakage of such mitigators is bounded by $N \log(T + 1)$.

3.4 Bounding leakage

Using Definition 2 and Equation 1 we may derive a leakage bound for epoch-based mitigation.

$$N \log(T + 1) + \sum_{j=1}^{N} \log \Lambda_j \le B(T) \tag{2}$$

Furthermore, if we consider mitigators where the transition variations are fixed—that is, there is $\lambda_{\max} \ge \log \Lambda_j$ for all j—then the leakage bound criterion for such mitigators can be expressed as a bound on the number of epochs.

$$N \le \frac{B(T)}{\log(T + 1) + \lambda_{\max}} \tag{3}$$

Define the *deferral point* D_N for epoch N to be the solution to the equation $N = B(T)/(\log(T + 1) + \lambda_{\max})$. The importance of D_N is that until D_N there must be at most N epochs; that is, the start of the $N+1$-th epoch has to be deferred until D_N. Because the $N+1$-th epoch starts with the misprediction at the N-th epoch, this leads us to the only security constraint for the choice of schedule S_N. Namely, for all events i, we should have $\forall\, N \ge 1\, .\, S_N(i) \ge D_N$, and consequently,

$$\forall N \ge 1\, .\, S_N(1) \ge D_N \tag{4}$$

Example.

Consider the basic scheme from Section 2.3, which has prediction function $S_N(i) = [t + i \cdot 2^{N-1}]^+$. For this scheme, we have $\lambda_{\max} = 0$. Consider bound $\log^2(T + 1)$, which leads to deferral points D_N for N-th epoch $D_N = 2^N - 1$. The leakage bound requires $S_N(1) \ge D_N$. Since in the basic scheme, the N-th epoch starts at 0 for $N = 1$, and at least at time $\sum_{i=0}^{N-2} 2^i$ for all $N \ge 2$, therefore the leakage bound follows from $S_N(1) = 0 + 2^0 = 1 \ge 2^1 - 1 = D_N$ for $N = 1$, and $S_N(1) \ge \sum_{i=0}^{N-2} 2^i + 2^{N-1} = 2^{N-1} - 1 + 2^{N-1} = 2^N - 1 = D_N$ for all $N \ge 2$.

Figure 2 shows the deferral points for the basic scheme from Section 2. Here the bound is $B(T) = \log^2 T$, $\lambda_{\max} = 0$, and the deferral points correspond to the intersections of the curves $N \log T$ with the bound curve.

Adaptive mitigators.

When a misprediction does not occur for a sufficiently long time, the difference $B(T) - N \log(T + 1) - \sum_{j=1}^{N} \log \Lambda_j$ may allow an extra epoch transition. Say that an epoch transition is *adaptive* when it is initiated by the mitigator rather than by a misprediction. Equations 2 and 3 can also be used to design criteria for adaptive transitions. In particular, an adaptive transition is secure when

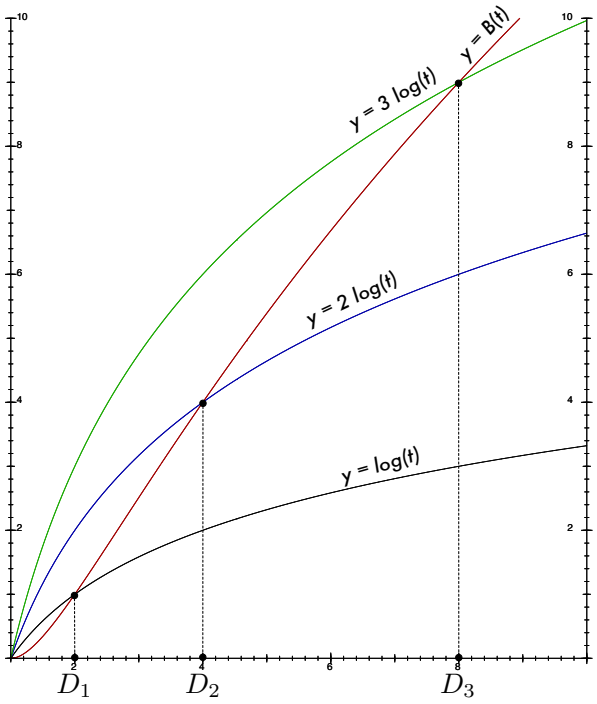


Figure 2: Target bound, capacity approximation for individual epochs, and deferral points

$$B(T) - N\log(T+1) - \sum_{j=1}^{N} \log \Lambda_j \geq \log(T+1) + \log \Lambda_{N+1}$$

Here Λ_{N+1} is the number of possible new predictions for epoch number $N+1$.

One use for adaptive transitions is to help reduce the size of the event buffer. Past deferral points, the mitigator can choose to release more than one event from the buffer. The number of choices for how many events can be flushed from the buffer then contribute to Λ_N for the mitigation scheme at that deferral point. Prudent mitigator design probably avoids completely emptying the buffer, since an empty buffer may risk an unpredicted miss.

A second example of using adaptive transitions to improve performance is given in Section 4.1.

On the choice of bound functions.

Because the epoch-based mitigation scheme is parametric on the choice of the bound function $B(T)$, we briefly discuss possible choices for practical bounds.

Recall that we assume the number of processed events in an epoch may leak information. Under this assumption, the most draconian bound possible is $\log T$. Enforcing such a bound effectively restricts output to a single epoch for the entire run of the program. In case of a misprediction, all subsequent events would have to be delayed until the end of the program.

Our simple and adaptive mitigators use the polylogarithmic bound $\log^2 T$, which appears to make a reasonable trade-off between performance and security. However, as this section illustrates, even with this more relaxed bound, the distance between deferral bounds increases exponentially.

A third choice corresponds to a larger, more permissive, class of bounds such as $kT + \log^n T$, for $n \geq 2$ and small (or zero) k. We have not explored such bounds in this work, though it is possible that a linear, albeit slowly growing, bound may be useful in bringing deferral points closer in practice.

3.5 Mixing storage and timing

A variety of information flow control techniques have been developed for controlling leakage through storage channels. We can now show that these techniques combine well with timing mitigation.

We use the information theoretic measure of mutual information, to measure leakage. Given random variables A and B, their mutual information $I(A;B)$ is the information that A conveys about B, and vice versa. It is defined as $I(A;B) = H(A) + H(B) - H(A,B)$, where the function H gives the entropy of a distribution. Note that the entropy of a variable with n possible values is maximized when all n outcomes are equally probable, in which case it is $\log n$ bits.

Assume X is a random variable that corresponds to secret input, Y is a random variable that corresponds to the storage channel, and Z is a random variable that corresponds to the timing channel. The amount of information that the attacker gains by observing both storage and timing channel is the mutual information between the secret and the joint distribution of Y and Z: that is, $I(X;Y,Z)$. Similarly, the amount of information that the attacker gains by observing just the storage channel is $I(X;Y)$.

The following easy theorem says that the information leaked by the combination of timing and storage channels is bounded by the information leaked by the storage channel, plus the maximum information content of the timing channel. The proof is in the appendix.

THEOREM 1 (SEPARATION OF STORAGE CHANNEL).

$$I(X;Y,Z) \leq H(Z) + I(X;Y)$$

A symmetric theorem can be stated for the timing channel, but seems less useful because of the difficulty of estimating $I(X;Z)$. A direct corollary to this theorem is that if the system enforces noninterference [12, 24] on the storage channel, the total secret information leaked from the system is bounded by the entropy of the timing channel.

3.6 Input

Event sources often communicate with the external world by accepting input, and block waiting for input when no input is available. Let us assume that the timing of input does not contain sensitive information, or at least that it is the responsibility of the input provider to control the input timing channel. The time spent by a computing system waiting for input clearly does not communicate anything about its internal state provider. Therefore, the system comprising the event source and mitigator should not be penalized for time spent blocked waiting for input. For the purposes of mitigation, the clock controlling the scheduling of slots can be stopped while the event course is blocked waiting for input. This refinement is particularly helpful when the event source is a service whose service time does not fluctuate much.

3.7 Leakage with beliefs about execution time

Finally, we consider a particular case of *server applications* that handle client requests. In this special case, a tighter, albeit probabilistic, bound on leakage can be established than is possible with the general framework presented thus far.

For many real applications that handle sequential client requests, such as RSA encryption and simple web services (see Section 4.4),

execution times fall within a narrow range, regardless of the values of secrets. We show that under the assumption that the distribution of execution times is approximately as expected, expected mitigated leakage can be given a tighter bound than $O(\log^2 T)$.

Suppose that with probability at least p, the execution time for a single request is at most T_{big}. That is, the adversarial insider controls execution time but cannot make the probability of exceeding T_{big} greater than $1 - p$. For some computations, such as blinded cryptographic operations on sufficiently isolated computers, p can be gained by sampling with randomly generated inputs. Given T_{big}, a corresponding number of epochs N_{big} can be calculated, giving the number of transitions that must occur before executions of length T_{big} are possible. For instance, in the basic doubling scheme, $N_{\text{big}} = \lceil \log(T_{\text{big}}) \rceil$. Under these assumptions, expected leakage $L(N_{\text{big}}, T)$ is derived using conditional entropy:

$$L(N_{\text{big}}, T) = p \cdot \log \mathbf{M}(T, N_{\text{big}}) + (1 - p) \cdot \mathbf{M}(T, N)$$

where, as before, $\mathbf{M}(T, N_{\text{big}})$ is the bound on the number of possible variations of a mitigator when N is at most N_{big}.

Example.

For the basic doubling scheme, given T_{big}, we know that $N_{\text{big}} \leq \lceil \log(T_{\text{big}} + 1) \rceil$. Using the formula for $\mathbf{M}(T, N)$, we can derive

$$L(N_{\text{big}}, T) \leq p \cdot (N_{\text{big}} \cdot \log(T + 1) - N_{\text{big}}(N_{\text{big}} - 1)/2) + \frac{1 - p}{2}(\log^2(T + 1) + \log(T + 1))$$

4. ADAPTIVE MITIGATION RESULTS

Some simple experiments with predictive mitigation help us understand how the mitigator can converge on the right separation between slots.

4.1 Convergence

Buffering source events helps prevent slowing down the event source and absorbs temporary variations in event rate. However, it is undesirable for the buffer to grow too large, because it increases latency. If the buffer fills, the event source must be paused to allow the buffer to drain. We would like to avoid significantly pausing well-behaved applications, because pauses could disrupt their functionality.

In this part, we focus on a simplified event source, and propose one way to add adaptive transitions to the basic mitigation mechanism of Section 2.3. This particular design typically allows the quantum converge to the event rate, while still keeping the information leakage lower than the desired bound. Empirical results demonstrate the convergence of the mitigator in face of many different event rates. Although currently our solution is restricted to certain input event patterns, the experiment suggests that adaptive, epoch-based mitigation may be practical for different applications.

Suppose we use the simple mitigation mechanism with constant-quantum positive predictions for every epoch. Consider an event source that generates events at a constant rate, say 1 event per every 8 seconds; call the interval between events the *event interval*. When the quantum of the mitigation system is higher than the event interval—say, 10 seconds—the mitigator begins accumulating events in its buffer queue. Eventually, an increase in the buffer size may reduce both the latency and throughput of the mitigator. On the other hand, if the quantum is smaller than the event rate—say, 6 seconds—then the buffer quickly drains, causing unwanted epoch transitions.

Therefore, designing an adaptive mitigation scheme that can converge roughly to the event interval of the source system is important for practical applications.

4.2 Assumptions

We now show that adaptive mitigation can work for relatively well-behaved event source. To capture the behavior of an event source that generates events at some average rate but with local variation around that rate, we work with an event source that generates one event at a random point during each fixed interval. It is easy to see the optimal prediction for an event source of this type is the one whose constant quantum matches the average interval between events.

The basic intuition behind the construction of the adaptive mitigation mechanism is that the size of the buffer indicates how the quantum should be adjusted. A quantum that is too large causes the buffer to grow large; a quantum that is too small causes the buffer to empty. Both of these conditions can be taken into account by the mitigator.

4.3 An adaptive mitigation heuristic

Following the idea of adaptive mitigation from Section 3.4, we heuristically extend the basic mitigator of Section 2 to adjust future schedules based the buffer size. There is no reason to believe that the particular mechanism is optimal; we describe this mechanism as a way of illustrating what is possible with adaptive mitigation.

In this mitigator, an adaptive epoch transition happens when both of the following conditions hold:

1. the size of the buffer queue is increasing, and
2. the mitigator would meet the leakage bound even if it transitioned into a new epoch.

Note that condition (1) here is specific to the design of the current mitigator, while condition (2) is a necessary condition for all adaptive transitions, as described in Section 3.4.

The adaptive mitigation heuristic works as follows. It doubles the quantum on each miss transition, which lets the quantum quickly approach the event interval. Next, the mitigator adjusts the quantum closer to the event interval by raising or lowering the quantum deterministically at every adaptive transition. The current quantum ideally fluctuates around the desired quantum and finally converges to it. We constrain the mitigator to have a deterministic reduction rate, enabling a deterministic (and small) bound on possible schedule functions.

This scheme uses reduction rates that regulate how quantum size is adapted. We denote reduction rates by r_j, where j ranges from 1 to 9, such that $r_1 = 0.95$, $r_2 = 0.9$, ..., $r_9 = 0.55$. Note that the number of reduction rates and the corresponding values for this experiment have been derived empirically, based on the experimental results, reported in Section 4.4.

The mitigator has an internal state, which is a pair (q, j). Here q is current quantum and j is the current reduction rate. Call the condition that guards when an adaptive transition may be done an *adaptive condition*; the next state (q', j') is computed at a transition point and is derived as follows:

$$(q', j') = \begin{cases} (q/2r_j, \text{next } j) & \text{if adaptive condition holds} \\ (2q \cdot r_j, \text{next } j) & \text{if miss occurs} \end{cases}$$

where function next specifies the choice of the next reduction rate

$$\text{next } j = \begin{cases} j + 1 & \text{when } j < 9 \\ 5 & \text{when } j = 9 \end{cases}$$

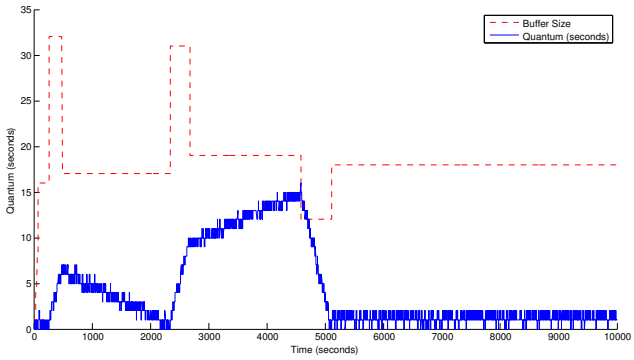


Figure 3: Adaptive mitigation with average interval of 18 sec.

Using the new state, the schedule for the next epoch is computed as $S_N(i) = [\tau + i \cdot q']^+$.

According to our discussion in Section 3.4, since the multiplier rates are deterministic, and there are only two possible transitions (speed up or slow down), we have $\lambda_{max} = 1$. If we set the total information leakage $B(T)$ to be $\log^2(T+1)$, the number of transitions must be no more than $(\log^2(T+1))/(\log\ (T+1)+1)$ as derived from the leakage bound criterion in Section 3.4. Adaptive transitions are not allowed if this constraint is not met.

4.4 Empirical results

Figure 3 illustrates how the quantum converges to the event interval through the adaptive mitigation mechanism when the event interval is 18 seconds. In this figure, the quantum is indicated by the dashed line and the buffer size is represented by the solid line. Initially the mitigator doubles the quantum quickly to 32, and then lowers the quantum because the queue size has grown, around the 350-second point. Then, the queue slowly drains because the quantum is smaller than the event interval. When the queue empties around 2000 seconds, the quantum is raised again. After several adjustments, the quantum finally converges to 18 at around 5000 seconds and stays constant thereafter. Once converged, the queue size remains small (around 2–3), ensuring low latency.

While perfect convergence is not required for this scheme to be useful, we tested the convergence with event intervals ranging from 1 second to 100 seconds, with the results shown in Figure 4. Each dot represents the final quantum arrived at by the mitigation system after different total run times. Three curves are shown, one for the final quantum after 10000 seconds, one for after 100000 seconds, and one for after 1000000 seconds. The plot shows that the adaptive mitigation heuristic converges closely to the event interval in most cases. However, there are certain cases where convergence never occurs, such as at an event interval of 42; here, the mitigation system loops among five values close to 42. The current set of reduction rates were chosen in a largely ad hoc fashion; we leave finding an optimal set for a broad class of event sources to future work.

4.5 Composing mitigators

Figure 5 illustrates convergence of composition of two adaptive mitigators. Here the first mitigator processes events received from a source system with an 18 sec. event interval. The second mitigator processes events that it receives from the first one. The lines on the graph illustrate the change of quantum values in each mitigator. Based on the similar experiments for other event intervals, we observe that composed mitigators converge in most cases. We leave identifying necessary and sufficient conditions for convergence to future work.

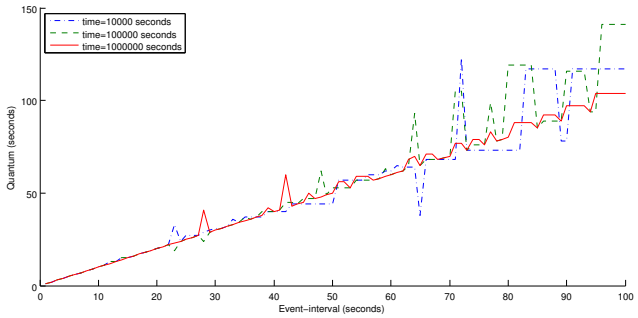


Figure 4: Convergence with different event intervals

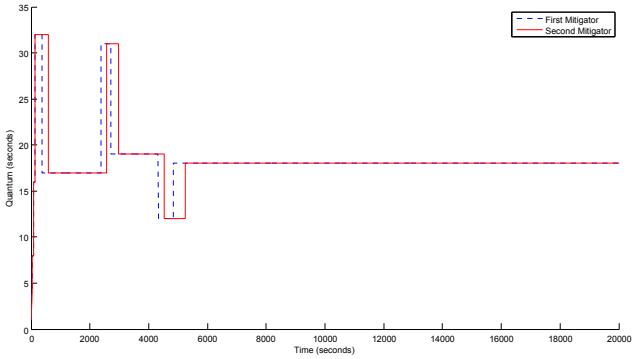


Figure 5: Convergence of composition of mitigators with average interval of 18 sec.

5. APPLICATION-LEVEL EXPERIMENTS

We evaluated the effectiveness of the basic timing mitigation mechanism on two published timing channel attacks: RSA timing channels [4] and remote web server timing channels [2].

Both experiments show that the basic mitigation mechanism of Section 2.3 can successfully defend against these timing channel attacks, although with a latency penalty.

5.1 RSA

To demonstrate the effectiveness of timing mitigation, we applied it to OpenSSL 0.9.7, a widely used open-source SSL library that was shown to be vulnerable to RSA timing channel attacks [4]. The results show that timing mitigation eliminates the time difference targeted by RSA timing channel attack, making this attack infeasible.

5.1.1 Experiment setup

The experiment was performed on OpenSSL 0.9.7. This version was used because it is the same version shown to be vulnerable by Brumley et al, and by default it does not use blinding to prevent timing channels. Measurements were made on a 3.16GHz Intel Core2 Due CPU, with 4G of RAM, using GCC 4.4.1. The attacker continuously asks the target to decrypt a message and records all decryption times, starting a new decryption request whenever the

last one is done. The Intel CPU cycle count obtained using the `rdtsc` instruction provided a precise, accurate clock.

5.1.2 Attack strategy

We used the timing channel attack strategy proposed in [4] for this experiment, attacking RSA keys with 1024 bits. Instead of trying to get the secret key directly, this attack targets the smaller factor of N used in RSA key generation. More specifically, the attacker attacks q, where $N = pq$ with $q < p$. Once q (512 bits for a 1024-bit key pair) is released, the attacker can easily derive the secret key by computing $d = e^{-1} \mod (p-1)(q-1)$.

The attack works by learning a bit of q at a time, from most significant to least. In each request, the attack generates two guesses (512-bit numbers) and records the decryption time for each guess. To set up, the attacker guesses the first 2–3 bits of q by trying all possible combinations (feeding rest of the bits as 0), and plots all decryption times with x-axis of guesses. The first peak in the graph corresponds to q. Once the attacker has recovered the top $i-1$ bits of q, two new guesses g_1 and g_2 are generated, where

1. g_1 has the same top $i-1$ bits as q and the remaining are zero.
2. g_2 differs from g_1 only at the i^{th} bit, by setting it to 1

The attacker then computes $u_{g_1} = g_1 R^{-1} \mod N$ and $u_{g_2} = g_2 R^{-1} \mod N$ (where R is some power of 2 used in Montgomery Reduction), and measures the time to decrypt both u_{g_1} and u_{g_2}. Denote by Δ the difference between these two decryption times.

The goal of this RSA timing channel attack is to find a 0–1 gap when a certain bit of q is 0 or 1. More specifically, when the i^{th} bit of q is 0, the decryption time difference Δ will be large, otherwise small. So the attacker wins by analyzing the significance of 0–1 gap to get all bits of q. Actually, after recovering the most significant half of the bits of q, attacker can use Coppersmith's algorithm [8] to recover the rest of the bits. So we only show the 0–1 gap for the first 256 bits of q in this experiment.

5.1.3 Parameter choices

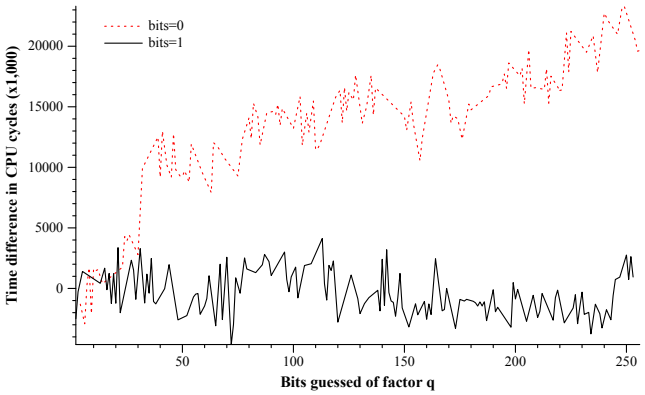
To overcome the effects of a multi-user environment, multiple decryptions for same guesses are necessary to cancel out the timing differences. Experimentally, we found the median time of 7 samples gives a reliable decryption time with very small variation, so this is the sample size used hereafter.

Measuring the decryption time for $n+1$ guesses ranging from g, $g+1, \ldots, g+n$ can make the 0–1 gap more significant, and thus brings more confidence in the guess, though at a computational cost to the attacker [4]. We chose 600 as the value for n, because it was enough to gain a significant 0–1 gap in most cases.

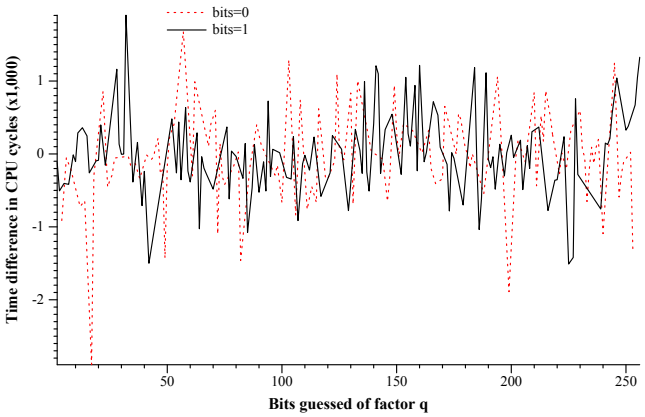
5.1.4 Timing mitigation of RSA Attack

Since the 0–1 gap does not depend on any specific key [4], we used a randomly generated 1024-bit key for our experiment. Figure 6(a) shows the result without any timing mitigation mechanism. The dotted line is the zero–one gap when the corresponding bit of q is 1, and the solid one is when the bit is 0. It is easy to see that an attacker can infer certain bits of q by observing this zero–one gap, especially when guessing a bit whose position is larger than 30. For bit indices less than 30, it is possible to increase the 0–1 gap by calculating a larger neighborhood set, with more cost to the attacker.

On the other hand, Figure 6(b) shows a RSA decryption process with the simple timing mitigation mechanism we proposed in Section 2. The timing channel attack on RSA is defeated because



(a) Without mitigation



(b) With mitigation

Figure 6: Simple mitigation of the RSA timing attack

the two curves are indistinguishable regardless of which bit is being guessed: our timing mitigation scheme eliminates the 0–1 gap. The mitigation mechanism makes the time difference drop by four orders of magnitude, because the only source of time difference is the request time, which does not depend on the currently guessed bit.

5.1.5 Expected leakage

If we are willing to make assumptions about the distribution of encryption time, we can apply the method for estimating expected leakage that is discussed in Section 3.7. Using 1000 randomly generated inputs to estimate T_{big}, we find that 99% of them are handled within $1 * 10^8$ clock cycles, which is approximately $\frac{1\times10^8\times10^3}{3.16\times10^9} = 31.65$ ms (this is a 3.6GHz CPU). With an initial quantum of 1 ms, it is easy to see that $N_{\text{big}} = \lceil \log(31.65) \rceil = 5$. The leakage bound in this case is shown in Figure 7, topping out for practical purposes around 100 bits.

5.2 Timing attack on web servers

Web applications have been shown vulnerable to timing channel attacks, either by direct timing or cross-site timing. For instance, many web applications try to keep secret whether a given username is valid, by returning the same error message regardless of validity. They do this because learned usernames can be abused for spam, invasive advertising, and phishing. However, timing can expose username validity, because sites usually execute different code paths for valid and invalid user names [2].

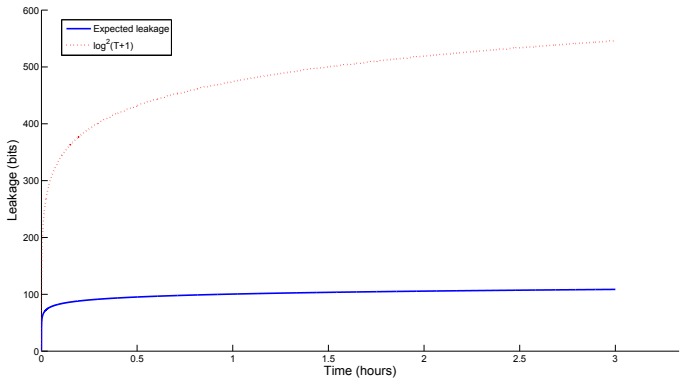


Figure 7: Expected leakage for RSA timing channel attack

We implemented a simple web server to expose this timing channel and applied our mitigation scheme to eliminate it. The result shows that our mitigation mechanism is also useful in the face of web applications, although with a latency cost.

5.2.1 Experimental setup

We built a small HTTP web service on Tomcat 5.5.28. It takes a username/password pair as a request and checks its validity. We randomly generate 10,000 username/password pairs, and store the username with a SHA-1 password hash of passwords into Berkeley DB (Java Edition, 4.0.92) [25]. This experiment is done between two computers connected by a campus network.

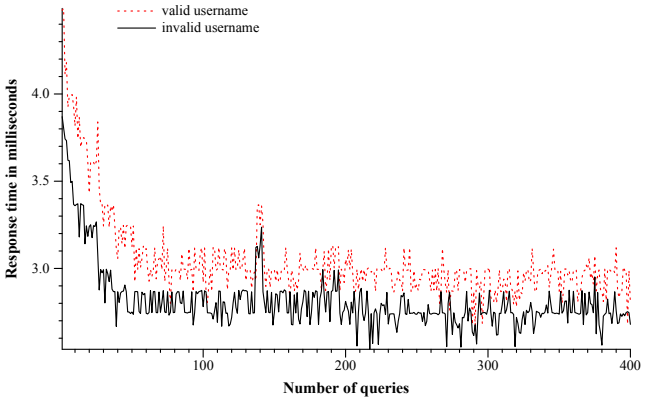
The login service proceeds as follows: first, it checks the database for validity of the given username. If the username is invalid, this server just returns an error message. Otherwise, the server computes the SHA-1 hash of the given password and checks if it matches the one stored in database. If the password does not match, the server returns the same error message as for an invalid username, to conceal username validity. This captures the essence of a login service. However, despite its simplicity, this service also exhibits a possible timing channel, because the computation of the SHA-1 hash depends on username validity.

To reduce network timing noise, we measure the query time 20 times for each username, and choose the smallest one as our sample. For each experiment, we randomly choose 400 valid usernames from a valid username list, as well as 400 randomly generated invalid usernames to determine the timing difference between them. As in the RSA experiment, we use a sequential attacker model, where the attacker issues a query immediately after the response. To make the difference more precise, we alternately issue valid and invalid queries.

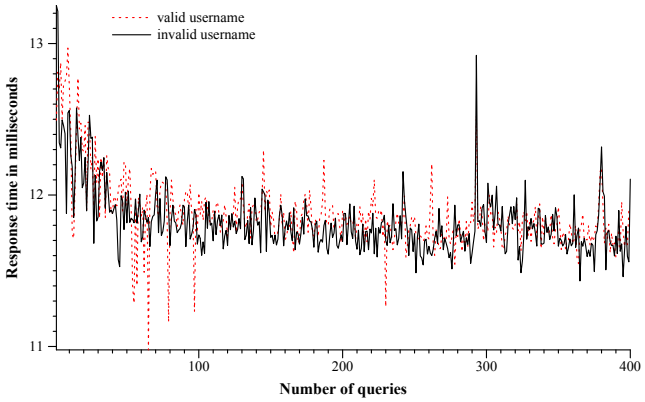
For the basic mitigation mechanism, instead of modifying the Tomcat source code, we wrap the `doGet` function in our login service servlet with code implementing the basic mitigation scheme, to control leakage of the time needed to look up and check the password. Because it is not implemented as part of Tomcat, this implementation cannot mitigate timing information communicated by web service setup time. However, the experiment still shows that timing mitigation can defend against this timing channel attack.

5.2.2 Results

Figure 8(a) shows how query time differs for valid and invalid usernames. Queries for valid usernames take significantly longer, so a timing channel attack is easy to mount. Web server setup adds about 1.5ms latency to queries in the beginning of the run, but the



(a) Without mitigation



(b) With mitigation

Figure 8: Simple mitigation of the web server timing attack

query time stabilizes after around 50 queries. An attacker could determine the validity of an arbitrary username with high confidence.

Figure 8(b) shows the response time with the basic mitigation mechanism. Since server replies only at the end of the current quantum, the time difference is independent of the validity of username. Close inspection of the results reveals that there is an initial 1.5ms timing difference that is not mitigated by our implementation. This timing difference is caused by the setup of the web service, rather than by the login service we mitigated, and underscores the importance of mitigating timing end-to-end rather than on individual system components.

Another observation not shown in the time difference graph of RSA experiment is that our simple timing mitigation mechanism also adds a latency penalty to the web service, since the service time is unified to the closest power of 2 of the largest service time. This latency can be seen in Figure 8(b), where mitigation is seen to add about 9 ms latency.

5.2.3 Expected leakage

We applied the expected-leakage approach of Section 3.7 to the web service. Using 1000 random requests, we determined that 99% of them are below 8 ms. Replacing $T_{\text{big}} = \lceil \log 8 \rceil$ and $p = 0.99$ with these two numbers, the expected leakage for this application as shown in Figure 9, with $q_0 = 1$ ms. Clearly, the mitigated version leaks information slowly in practice.

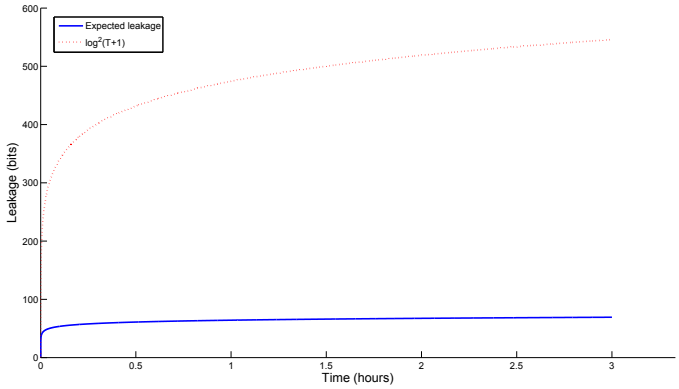


Figure 9: Expected leakage for web server timing attack

6. RELATED WORK

Timing channels have been widely studied in the literature. We briefly explore related work below.

Cryptographic side-channels.

One major motivation for controlling timing channels is the protection of cryptographic keys against side-channels arising from timing cryptographic operations. A variety of attacks that exploit timing side-channelshave been demonstrated [4, 17]. Cryptographic blinding [5, 17] is a standard technique for mitigating such channels. In recent works, Köpf et al. [18, 19] utilize blinding with quantization (referred in there as *bucketing*) to derive tight bounds on leakage of cryptographic operations.

Quantitative information flow.

We advocate a quantitative approach to controlling information flow through timing channels. Like much other work on quantitative information flow [6, 23, 18, 19] we draw on information theory to obtain bounds on leakage. Millen [24] first observed that noninterference implies zero channel capacity between high and low. DiPierro et. al [28] quantify timing leaks in a language-based setting. Epoch-based mitigation is similar in spirit to Mode Security [3] which reduces covert channels to changes in modes. Unlike Mode Security, we also account for leakage within epochs, via a combinatorial analysis.

Detection of timing attacks.

Some prior work on timing channels has focused on detecting the perturbation in the distribution of times introduced by timing attacks [10]; however, stealthy timing attacks have been demonstrated [21, 22].

Mitigation of timing attacks.

Giles and Hajek present a comprehensive study of timing channels [11] in which packet arrival is represented by continuous or discrete waveforms. Similarly to us, they employ periodic quantization. However, because of the constant periods, the reduction of the timing channel bandwidth is only linear. Another difference lies in the semantics of buffer bounds: while they assume that a jammer has to release a packet from the queue when a buffer is full, our mitigators block the input source.

One prior approach to timing channel mitigation is adding noise to timing measurements. There are two ways to do this. First, we can add random delays to the time taken by various operations, which reduces the bandwidth of the timing channel, as in [14, 11]. Adding random delays sacrifices performance, and it does not asymptotically eliminate timing channels, since the noise can be eliminated to whatever degree is desired by averaging over a sequence of identical requests. Methods for creating covert timing robust against added noise have been demonstrated [21].

A second approach to mitigation, also used in [14], is that programs that read clocks are given results with random noise. This method only applies to internal timing channels that are based on reading clocks directly.

Wray [32] views every covert channel that originates from comparing two clocks as a timing channel. In this light, we focus on the channels that arise from comparing timing of the events to external reference clock that is not modulated by the attacker. Our results of Section 3.5 can be interpreted as mixing external timing channels and all other covert channels.

The line of work on NRL Pump [15, 16] addresses timing channels that arise when high confidentiality processes acknowledge receipt of messages from low confidentiality processes.

In a language-based setting, it is possible to reason about ways the program can measure time, and language-based methods have been proposed for controlling internal timing channels by analysis [33] and by transformation [1, 29]. Coppens et al. [7] explore automating compiling techniques to defend against timing-based side-channel attacks on x86 processors. Language-based methods for mitigating general external timing channels have also been proposed, but rely on unrealistic assumptions. For example, Agat's work [1] ignores the effect of the code cache on timing, and is limited to programs that lack loops and recursion. Shroff and Smith lift some of Agat's limitations [31], but at the cost of possibly disrupting computations.

7. CONCLUSION

This paper has introduced a new class of schemes for mitigating timing channels for general computer systems. The key intuition is that the timing mitigator can often predict the future availability of events to deliver. Mitigator predictions divide time into epochs. When a prediction fails, a new epoch begins and some information is leaked. The mitigator is able to track the amount of information leaked at each epoch transition and to enforce whatever leakage bound has been specified. When the information bound permits, the mitigator can also adaptively start a new epoch for improved performance.

This paper has identified the key conditions that an epoch-based mitigator must satisfy, and described some useful adaptive mechanisms. However, there is no doubt more work to be done on understanding the space of epoch-based timing mitigators. The problem of generating schedules of predictions for these mitigators, particularly for various classes of applications, appears interesting.

This paper has considered combining timing mitigation with other mechanisms for controlling information flow through storage channels, and shown that it is possible to conservatively bound the capacity of the combined channel by building on the analysis of timing channel capacity given here. However, we have not yet implemented such a combined mechanism for information flow control; this is clearly a useful future direction. Exploring epoch-based timing mitigation in real-world systems is an obvious next step.

8. REFERENCES

[1] J. Agat. Transforming out timing leaks. In *Proc. 27th ACM Symp. on Principles of Programming Languages (POPL)*, pages 40–53, Boston, MA, Jan. 2000.

[2] A. Bortz and D. Boneh. Exposing private information by timing web applications. In *Proc. 16th Int'l World-Wide Web Conf.*, May 2007.

[3] R. Browne. Mode security: An infrastructure for covert channel suppression. In *IEEE Symposium on Research in Security and Privacy*, pages 39–55, May 1994.

[4] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, Jan. 2005.

[5] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[6] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.

[7] B. Coppens, I. Verbauwhede, K. D. Bosschere, and B. D. Sutter. Practical mitigations for timing-based side-channel attacks on modern x86 processors. *IEEE Symposium on Security and Privacy*, pages 45–60, 2009.

[8] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4), Dec. 1997.

[9] R. G. Gallagher. Basic limits on protocol information in data communication networks. *IEEE Transactions on Information Theory*, 22(4), July 1976.

[10] S. Gianvecchio and H. Wang. Detecting covert timing channels: an entropy-based approach. In *CCS '07*, Oct. 2007.

[11] J. Giles and B. Hajek. An information-theoretic and game-theoretic study of timing channels. *IEEE Transactions on Information Theory*, 48(9):2455–2477, 2002.

[12] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, Apr. 1982.

[13] D. M. Goldschlag. Several secure store and forward devices. In *CCS '96*, pages 129–137, Mar. 1996.

[14] W.-M. Hu. Reducing timing channels with fuzzy time. In *IEEE Symposium on Security and Privacy*, pages 8 – 20, 1991.

[15] M. H. Kang and I. S. Moskowitz. A pump for rapid, reliable, secure communication. In *CCS '93*, pages 119–129, Nov. 1993.

[16] M. H. Kang, I. S. Moskowitz, and S. Chincheck. The pump: A decade of covert fun. In *ACSAC '05*, pages 352–360, 2005.

[17] P. Kocher. Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO'96*, Aug. 1996.

[18] B. Köpf and M. Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *2009 IEEE Computer Security Foundations*, July 2009.

[19] B. Köpf and G. Smith. Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In *2010 IEEE Computer Security Foundations*, July 2010.

[20] B. W. Lampson. A note on the confinement problem. *Comm. of the ACM*, 16(10):613–615, Oct. 1973.

[21] Y. Liu, D. Ghosal, F. Armknecht, A. Sadeghi, and S. Schulz. Hide and seek in time—robust covert timing channels. In *ESORICS*, 2009.

[22] Y. Liu, D. Ghosal, F. Armknecht, A. Sadeghi, S. Schulz, and S. Katzenbeisser. Robust and undetectable steganographic timing channels for i.i.d. traffic. In *Information Hiding 2010*, June 2010.

[23] G. Lowe. Quantifying information flow. *Proc. IEEE Computer Security Foundations Workshop*, June 2002.

[24] J. K. Millen. Covert channel capacity. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, 1987.

[25] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *Proc. USENIX Annual Technical Conference*, 1999.

[26] D. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of AES. *Topics in Cryptology–CT-RSA 2006*, Jan. 2006.

[27] M. Padlipsky, D. Snow, and P. Karger. Limitations of end-to-end encryption in secure computer networks. Technical Report ESD TR-78-158, Mitre Corp., 1978.

[28] A. D. Pierro, C. Hankin, and H. Wiklicky. Quantifying timing leaks and cost optimisation. *Information and Communications Security*, 2010.

[29] A. Russo, J. Hughes, D. Naumann, and A. Sabelfeld. Closing internal timing channels by transformation. In *Proc. 11th Annual Asian Computing Science Conference (ASIAN)*, 2006.

[30] G. Shah, A. Molina, and M. Blaze. Keyboards and covert channels. *Proc. 15th USENIX Security Symp.*, Aug. 2006.

[31] P. Shroff and S. F. Smith. Securing timing channels at runtime. Technical report, The John Hopkins University, July 2008.

[32] J. C. Wray. An analysis of covert timing channels. In *IEEE Symposium on Security and Privacy*, pages 2–7, 1991.

[33] S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *Proc. 16th IEEE Computer Security Foundations Workshop*, pages 29–43, Pacific Grove, California, June 2003.

APPENDIX

A more precise bound on leakage of the basic scheme.

This derivation is based on the fact that each possible string can be determined by the placement of the misses, that is, the locations of "–" in the string. For m misses in time T, there are at most $\binom{T}{m}$ different strings. So

$$\text{All possible strings} \leq \sum_{m=0}^{\log T} \binom{T}{m} \leq (\log T + 1)\binom{T}{\log T} \leq (\log T + 1)\frac{T^{\log T}}{(\log T)!}$$

Thus, the leakage can be no more than $\log(\log T + 1) + \log^2 T - \log((\log T)!)$, and by Stirling's approximation,

$$\log((\log T)!) = \log T \log\log T - \log T + o(\log T)$$

So the whole leakage term is $O(\log T(\log T - \log\log T)))$.

Proof of Theorem 1.

We prove the theorem by using the definition of $I(X;Y)$ to show that the expression $H(Z) + I(X;Y) - I(X;Y,Z)$ is nonnegative.

$$\begin{aligned}
&H(Z) + I(X;Y) - I(X;Y,Z) \\
&= H(Z) + H(X) + H(Y) - H(X,Y) \\
&\qquad - H(X) - H(Y,Z) + H(X,Y,Z) \\
&= H(Z) + H(Y) - H(X,Y) - H(Y,Z) + H(X,Y,Z) \\
&\geq H(Z) + H(Y) - H(X,Y) - H(Y) - H(Z) + H(X,Y,Z) \\
&= H(X,Y,Z) - H(X,Y) \geq 0
\end{aligned}$$

□