Poster: Predictive Mitigation of Timing Channels for Interactive Systems

Danfeng Zhang (PhD student) Aslan Askarov (Postdoctoral Associate) Andrew C. Myers (Faculty)
Cornell University

I. Introduction

Timing channels remain a difficult and important problem for information security. The time at which a computing system performs some observable action such as sending a network packet can in principle encode an unbounded amount of information about what is happening inside the system. An adversary able to accurately measure this time may learn confidential information from this side channel (e.g., [1], [2], [3], [4]); an adversary able to influence this time may additionally use it as a covert channel to communicate confidential information (e.g., [5], [6], [7]).

This work presents a practical and general approach that can mitigate timing channels to asymptotically logarithmic leakage under reasonable assumptions. Our approach is based on our recent on *predictive mitigation* of timing channels [8]. We generalize predictive mitigation to a larger and important class of systems: systems that receive input request from multiple clients and deliver responses. The new insight is that timing predictions may be a function of any public information rather than being a function simply of output events. Based on this insight, a more general mechanism and theory of predictive mitigation becomes possible. This results in tight bounds on timing leakage and enables effective application of predictive mitigation to a wide range of web applications. With reasonable settings, the mechanism brings only about 30% latency overhead, while allowing at most 850 bits for 100,000 requests.

II. System Model

We consider a class of systems that accept input requests from a variety of clients and send back responses. Figure 1 illustrates how predictive mitigation works in such a system. Two central elements in this model are the *service* and the *timing mitigator*. The timing of events produced by the service is influenced by confidential information, and the adversary may be able to affect how confidential information influences timing, to use timing as a covert channel to learn that information.

The mitigator controls timing channel leakage by delaying the output events according to a pre-defined schedule. As long as the events arrive according to (or ahead of) the schedule, leakage must be low because the only information leaked is the number of events. We call a period when all predictions are met an *epoch*. When the service fails to behave according to the schedule, the mitigator records the *misprediction* and switches into a new schedule. The new schedule is chosen in such a way that the number of epochs grows slowly with time.

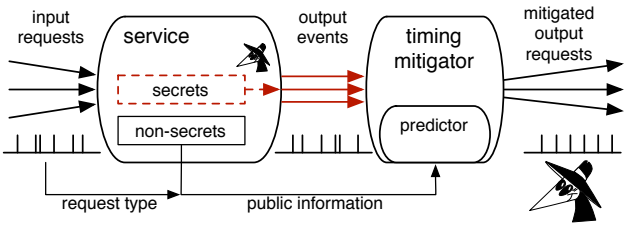

Figure 1. Predictive mitigation of an interactive system

III. Predictions for interactive systems

The more information is available to the mitigator, the more accurate are its schedules. The main idea of this work is to utilize public information about input timing, their request types and thread model to further improve accuracy of the predictions and the overall performance.

Schedules and predictions. We assume there is a correspondence between public inputs and public outputs; without loss of generality, we assume one-to-one mapping between input and output events.

We model the public inputs as a sequence of requests $(inp_1, r_1) \ldots (inp_i, r_i) \ldots$, where inp_i is the time of request i, and r_i identifies the type of the request. Request types capture what information about the request is public and can influence the mitigator. Let I_r be the number of mispredictions on request type r, and I be the collection of all such I_r's. Because I only changes when a misprediction happens, we use I formally as the epoch that it represents.

Assume that the predictions for epoch I are given by a function $p(I, r_i)$ that returns an expected bound on how long the it takes to compute an output for request type r_i in epoch I. The schedule for epoch I is as follows: if $S_I(0)$ is the start time of the epoch I, subsequent event i in epoch I is predicted to occur at time $S_I(i)$:

$$S_I(i) = \max[inp_i, \mathsf{avail}(I, i)] + p(I, r_i)$$

The two terms in the above expression correspond to the predicted start of the computation for event i and the predicted amount of time it will takes to compute the output, respectively. To predict the start of the computation for event i, we take the maximum time between when the input for that event is available and the time when the system is

predicted to be available to handle request i, denoted here by term $\mathsf{avail}(i)$.

Penalty policies. To show how predictions are defined, consider the "fast doubling" scheme which predicts computation time in the form of 2^n for some n. The choice of value of n for every request types determines the trade-off between performance and leakage.

We define the global index $G = \sum I$. A global penalty function, where $n = G$ for all request types enforces a tight leakage bound, but the performance can be bad. A local penalty function $n = I_r$ for type r gives good performance but leakage is linear to the number of request types.

We introduce a novel policy called *grace period* which gives a better trade-off. With an l-level grace period policy, each request type is given l tokens. Token of type r is reduced only when r meets a misprediction. When type r's token is not used up, r is penalized locally. That is, $n = I_r$. Otherwise, r is penalized globally. That is, $n = G$. Therefore the request types with regular timing behavior are not hurt by other "irregular" request types.

Leakage analysis. Now we analyze the leakage of l-level grace period policy with M inputs, R request types and elapsed time T. Careful analysis can bound the timing variation of possible outputs with $(M+1)^{(R-1)\cdot(l+1)+\log(T+1)+1}$. Taking the logarithm of the variation derived above, we can get the leakage bound in bits: $\log(M+1) \times ((R-1)\cdot(l+1)+\log(T+1)+1)$.

For most interactive systems, we can assume a worst-case execution time T_w, such as browser's timeout setting[1]. Given this constraint, similar derivation gives the bound of

$$\log(M+1) \times ((R-1)\cdot(l+1)+\log(T_w+1)+1)$$

Note that this is asymptotically $\log(M+1)$. This is a reasonably tight leakage bound for interactive systems.

IV. Experiments

As a part of our evaluation, we developed a mitigating HTTP proxy between client browser and real-world web applications to estimate the overheard of mitigating real-world applications. Figure 2 shows the average latency of load time for a selected web page and the associated bounds on the leakage, when the mitigator is used with 5-level grace period penalty policy. In this particular experiment, the selected web page results in 49 different requests to the server. Our evaluation considers different possibilities of grouping these requests.

1) TYPE/HOST: all URLs residing on the same host are treated as one request type.
2) HOST+URLTYPE: different request on the same host are predicted differently based on the URL type of the request. We distinguish URL types based on the file types, such JPEG files, CSS files and so on.
3) TYPE/URL: individual URLs are predicted differently in this case.

[1]We used 300 seconds in the experiments, which is the default setting of Firefox browser v. 3.6.12

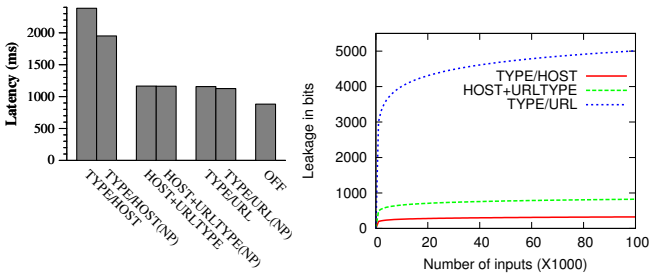

Figure 2. Latency and leakage bound for HTTP web page

With each setting, yet another option is to filter out some requests that do not leak crucial information. We use filtering images as another dimension (NP: no pictures) in this part.

Results. The results show that in the most restrictive TYPE/HOST case, the latency is almost tripled compared to the unmitigated case. Filtering out images reduces the latency in this case. HOST+URLTYPE and TYPE/URL options have similar latency results, with around 30% latency overhead. Moreover, filtering out images does not improve latency much for these cases.

From the security point of view, three options have 2, 7 and 49 request types respectively. The information leakage bound accordingly are shown in Figure 2, calculated by the leakage formula introduced before. The results show that HOST+URLTYPE provides a reasonable trade-off for request types selections: it has only around 30% latency overhead, while the information leakage in this setting is kept below 850 bits for 100,000 requests.

References

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

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

[3] D. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," *Topics in Cryptology–CT-RSA 2006*, Jan. 2006. [Online]. Available: http://www.springerlink.com/index/F52X1H55G1632L17.pdf

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

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

[6] H. Meer and M. Slaviero, "It's all about the timing..." in *Proc. Black Hat USA*, 2007.

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

[8] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in *ACM Conf. on Computer and Communications Security (CCS)*, 2010, pp. 297–307.