An Efficient Mitigation Method for Timing Side Channels on the Web

Sebastian Schinzel*

University of Mannheim, Laboratory for Dependable Distributed Systems

1 Introduction

Research has shown that timing side channels exist in web applications [1, 3, 5]. An obvious, but problematic, mitigation for timing attacks is to delay the execution time to the worst case execution time, so that all requests have the same response time. On the upside, this prevents timing attacks as there are no differences in the response time any more. On the downside, this approach has a negative effect on performance, which may render the approach useless for many practical systems.

In this extended abstract, we propose a new strategy to prevent timing attacks in web applications with little impact on performance. Our approach offers a provable security gain that can be freely traded for a performance decrease. We compare our approach to other strategies using two characteristics: firstly the added cost for an attacker to perform a side channel attack, and secondly the performance impact on the system.

The rest of this paper is organised as follows: In section 2, we describe timing side channel attacks on the web and review existing research for timing side channel mitigation. Section 3 provides a formal system model for the timing side channels we aim to mitigate. Section 4 compares existing timing side channel mitigation strategies with our own approach.

2 Timing Side Channels on the Web

A web application is vulnerable of timing side channels if the application leaks secret information in its response time behaviour. Take a login form as an example. The fact whether a given user account is registered is confidential information in the context of this application. On a failed login attempt, a web application should return the same error message independently if the user name or the password is wrong. If it returns different error messages, e.g. "*Wrong user name*" or "*Wrong password*", then the attacker learns from the error message if this user name belongs to a registered account. Even if the error messages hide this fact, the application could still leak information by different response times of login requests depending on whether the given user name exists.

Several authors researched techniques that can mitigate timing attacks on the web. Kocher [4] notes that making all operations take exactly the same amount of time is difficult. That is because it requires knowledge about the worst case execution time ($WCET$),

* Sebastian Schinzel was supported by Deutsche Forschungsgemeinschaft (DFG) as part of SPP 1496 "Reliably Secure Software Systems".

which is hard to estimate in non-real-time systems [6]. He states that adding a random delay will increase the number of necessary measurements that the attacker has to perform. It will therefore increase the cost for an attacker. However, it does not completely prevent timing leaks.

Bortz et al. conclude that fixing web server responses to a constant amount of time is insufficient as chunked encoding may leak information through inter-chunk timings [1]. Instead, they propose to fix the inter-chunk timings to a fixed value n. That way, if n is greater than the maximum time to prepare a chunk, then the measured timing will leak no information. If n is smaller than the chunk-preparation time, the system does leak information, but the attacker will get a lower resolution compared to a measurement without artificial delay. The measurements are therefore harder to conduct for the attacker. In their paper, they don't mention the performance impact introduced by their mitigation method.

Nagami et. al fix the execution time of only those processes that leak information through a timing side channel in order to limit the performance impact to selected processes [5]. As an example, they patched several web applications so that only the response time of the login process is fixed. Their approach is reactive as they focus on the prevention of already discovered side channels.

In the following, we formalise our model of side channel attacks against web applications.

3 System Model

In this paper, we analyse side channels that leak information about the existence of a given value. An example for this model is the login form of a web application that leaks information about the existence of a given user name through its timing behaviour.

We model such a process of a web application containing timing side channels as a deterministic function $f(U) \to T$. f takes a user-chosen value $u \in U$ and returns a deterministic timing delay $t \in T$. $S \subset U$ is a subset of all existing values and considered to be confidential. The following example illustrates the requirements that are necessary for f to form a timing side channel. Let $u_a \in S$ match a secret value and $u_b \notin S$ not match a secret value. f forms a timing side channel iff (if and only if)

$$\forall u_a \in S, u_b \notin S : f(u_a) \neq f(u_b)$$

We assume in our model that the attacker has perfect measurement conditions, i.e. there is no measurement error.

To decode the information leaked in the side channel, the attacker needs to know at least one u_0 and its affiliation with S, i.e. the attacker knows if $u_0 \in S$ or $u_0 \notin S$. The attacker then measures $f(u_0) = t_0$. Afterwards, he chooses a value u_1 and measures the timing t_1. Depending on $t_0 = t_1$ or $t_0 \neq t_1$, respectively, the attacker can infer whether $u_1 \in S$ or $u_1 \notin S$. Table 1 denotes the possible conclusions the attacker can draw, e.g. if $u_0 \in S$ and $t_0 = t_1$ then $u_1 \in S$, etc.

In the following section, we use this model to formalise two existing strategies and propose a new strategy.

	$t_0 = t_1$	$t_0 \neq t_1$
$u_0 \in S$	$u_1 \in S$	$u_1 \notin S$
$u_0 \notin S$	$u_1 \notin S$	$u_1 \in S$

Table 1. The conclusions an attacker can draw about u_1 from knowledge of u_0 and comparing t_0 to t_1.

4 Types of Mitigation

An intuitive approach to mitigate timing side channels is to hamper the attacker's ability to decode the side channel by adding an artificial delay to the execution time. We now analyse several strategies to delay the execution time and determine their ability to prevent timing attacks (cost for the attacker) versus their impact on system performance (cost for the system).

4.1 Fixing the Response Time

An intuitive mitigation for timing side channels is to remove the differences in the timing of responses [1, 3–5]. This can be done by delaying execution times $t = f(u)$ until a fixed time barrier t_f is reached. Figure 1 depicts this mitigation strategy. The attacker will then always measure t_f. The added delay t_d is calculated by $t_f - t$. Additional timing delays are strictly additive and we have to assume that $\forall t : t \leq t_f$ to make sure that all response times are equal.

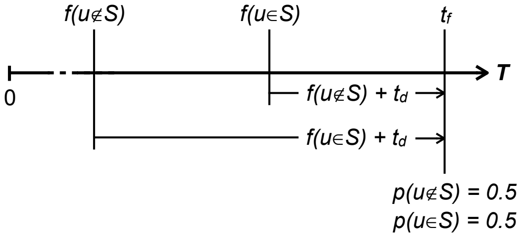

Fig. 1. Forcing all responses to a fixed response time prevents timing leaks but results in large performance penalty.

Let $u_a \in S$ and $u_b \notin S$. As shown in section 3, a necessary condition for timing side channels to exist is $f(u_a) \neq f(u_b)$. By fixing all response times to t_f, we get $f(u_a) = f(u_b) = t_f$. Thus, the necessary condition for timing side channels to exist is violated and the information leak is closed.

In real web applications, timing measurements are blurred significantly because of busy network conditions. To prevent timing side channels in these systems t_f needs to be at least as long as the worst case execution time ($WCET$), i.e. $t_f = max(f(U))$. Note that the maximum response time of web applications on the Internet is usually much larger than the average response time, because of its inherent highly skewed distribution [2]. Thus, degrading the system's performance to *WCET* for all requests is not acceptable in realistic systems. We therefore discard this strategy for further comparison.

4.2 Adding a Random Delay

We showed in section 3 that an attacker decodes a timing side channel by comparing the response time t_1 of a chosen value u_1 with the timing t_0 of a known value u_0. To make this decoding process more difficult for the attacker, one could add an additional random delay t_r to all response times [4]. $t_r \in T$ changes randomly for each response and is a uniformly distributed and randomly chosen number in the range of $0 \leq t_r \leq t_{max}$.

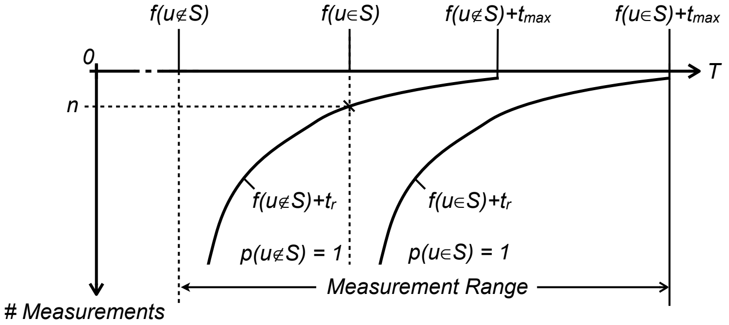

Fig. 2. Padding response times with a random delay forces the attacker to perform multiple measurements to estimate $f(u)$.

As a result, the attacker can only measure $f(u) + t_r$ and has to collect multiple values in order to learn the distribution of t_r and estimate the actual $f(u)$. Figure 2 shows the range of results that the attacker can measure. Given that the attacker conducts enough measurements, he can calculate $|(t_0 + t_r) - (t_1 + t_r)|$ which approximates $|t_0 - t_1|$. Thus, by performing multiple measurements for the same u, the attacker can approximate $f(u)$. Decoding the timing side channel is therefore costlier for an attacker when a random delay is included. The average performance reduction per request is $\frac{t_{max}}{2}$.

4.3 Adding a Fixed and Unpredictable Delay

Our system model in section 3 shows that an attacker decodes a timing side channel by comparing response times. As opposed to the random delay described in section 4.2, our idea is not to hamper the attacker's ability to measure t, but to limit the gained information from the comparison of $t_0 = f(u_0 \in S)$ and $t_1 = f(u_1 \notin S)$. We add a delay that is always the same for a given u and thus can not be factored out by multiple measurements as it is the case with a random delay. Formally, let $g(U) \rightarrow T$ be a deterministic function that takes a user-chosen value u and transforms it into a delay t_g

$$t_g \in T : t_{min} \leq t_g \leq t_{max} \wedge t_{max} > |f(u \in S) - f(u \notin S)|$$

For multiple u, g returns uniformly distributed delays that are not predictable by an attacker. Figure 3 introduces an algorithm for the function g. The algorithm builds around a cryptographic hash function h that hashes u. In order to make the resulting hash unpredictable for the attacker, we concatenate a secret key k to u before the hashing. We then perform a modulus operation to reduce the hash to the range $0 \leq t_g < t_{max}$. In practice, this algorithm produces uniformly distributed delays if the length of the hash is much larger than t_{max}.

```
function g(u):
    k := {secret key unknown to the attacker}
    t_g := h(u, k) mod t_max
    sleep_nano_seconds(t_g)
```

Fig. 3. Pseudo code describing the implementation of function g.

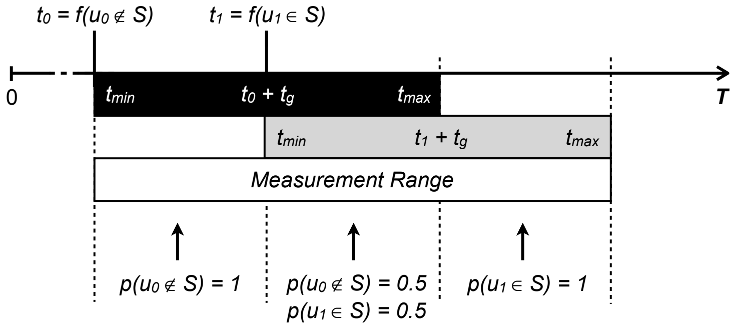

Fig. 4. Adding an unpredictable and fixed delay to responses protects an adjustable share of secret values from leaking through the side channel.

Figure 4 displays the three different areas of timing attack prevention. The left area ranges from t_0 to t_1. All t in this range leak information because all t are derived from $t_0 + t_g$, with $0 \leq t_g < t_1 - t_0$. As $t_0 + t_g < t_1$, any t in this area is derived from u_0.

The middle area in figure 4 ranges from t_1 to $t_0 + t_{max}$. All t in this area are either derived from $t_0 + t_g$, with $t_1 - t_0 \leq t_g < t_{max}$ or $t_1 + t_g$, with $0 \leq t_g < t_1 - t_0$. Any t in this area has an equal probability to be derived from u_0 or u_1. Consequently, there is no information leakage in this timing area as all t in this area are equally likely to be derived from u_0 or u_1, respectively. Formally, this area violates the necessary condition of timing side channels to exist as $\exists u_a \in S, u_b \in S : f(u_a) \neq f(u_b)$, which proves that there exists no timing side channel in this area.

The right area in figure 4 ranges from $t_0 + t_{max}$ to $t_1 + t_{max}$. Any t in this area leaks information because it is derived from $t_1 + t_g$ with $t_1 - t_0 \leq t_g \leq t_{max}$. Any t in this timing area is derived from u_1 as $t_0 + t_{max} < t_1 + t_g$.

Thus, it is desirable to increase the size of the middle area as much as possible. The share a of all possible user-chosen values U that should be protected from timing leaks is used to derive t_{max}, i.e.

$$t_{max} = \frac{|t_1 - t_0|}{1 - a} + t_{min}$$

As an example, an application needs to protect $a = 80\%$ of U from information leakage through the timing side channel. t_{min} is set to $0ms$ and $|t_1 - t_0|$ was measured to be $1ms$. The resulting t_{max} is $5ms$. The average performance impact of $g(U)$ is $\frac{t_{max}}{2} = 2.5ms$.

We believe that our mitigation method is more performance efficient than fixing the response time to WCET (see section 4.1) if the timing difference $|t_1 - t_0|$ is small compared to the variance of the measurements.

5 Conclusion

In this extended abstract, we formalised our idea for a performance efficient mitigation strategy for timing side channels in web applications. Our strategy works by delaying the response time of web applications, whereby the delay depends on the user-chosen value that was passed in the response. We showed that our strategy is guaranteed to protect a portion of the secret values from leaking through the side channel. The size of the portion is adjustable and can be traded with performance reduction.

In future work, we will implement our strategy in real-world web applications and validate it empirically. We will also apply our method to web servers using chunked encoding to mitigate the pitfalls described by Bortz et al. [1].

References

1. A. Bortz and D. Boneh. Exposing private information by timing web applications. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *WWW*, pages 621–628. ACM, 2007.
2. S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur*, 12(3), 2009.
3. E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *SIGSAC: 7th ACM Conference on Computer and Communications Security*. ACM SIGSAC, 2000.
4. P. C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In *CRYPTO: Proceedings of Crypto*, 1996.
5. Y. Nagami, D. Miyamoto, H. Hazeyama, and Y. Kadobayashi. An independent evaluation of web timing attack and its countermeasure. In *Third International Conference an Availability, Reliability and Security (ARES)*, pages 1319–1324. IEEE Computer Society, 2008.
6. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3), Apr. 2008.