Stochastic Models for Remote Timing Attacks

Simone Bozzolan
Università Ca' Foscari Venezia
simone.bozzolan@unive.it

Diletta Olliaro
Università Ca' Foscari Venezia
diletta.olliaro@unive.it

Stefano Calzavara
Università Ca' Foscari Venezia
stefano.calzavara@unive.it

Andrea Marin
Università Ca' Foscari Venezia
marin@unive.it

Gianfranco Balbo
Università di Torino
balbo@di.unito.it

Matteo Sereno
Università di Torino
matteo.sereno@unito.it

Abstract

In this paper, we present the first remote timing attack based on formal stochastic models. Our attack uses queuing models from the field of performance evaluation to estimate the service times of different classes of network requests. By using Bayesian statistics, we then identify opportunities for remote timing attacks by answering the following inverse question: what is the probability that a given network request belongs to a target class, given an estimate of its service time? Our experimental evaluation on popular web applications and websites shows that our investigation is not just a theoretical exercise, because our attack outperforms existing empirical approaches in terms of standard performance figures. We believe that the formal foundations put forward in this paper can be successfully applied to the creation of principled remote timing attacks which are more effective, because better equipped to deal with the complexity of the problem they are trying to solve.

Keywords

web privacy, queuing theory, remote timing attacks

1 Introduction

Timing attacks abuse timing information to infer private information about a target system. Timing attacks on the Internet are called *remote* timing attacks and have been proven to have significant practical consequences [7]. For instance, Brumley and Boneh showed that an attacker could measure the response-time variances of a remote web server to reconstruct its RSA private key and thus break cryptography [5]. In the privacy setting, Felten and Schneider used a cache-based timing attack to track web users [8]. Their idea is that once a user visits a static page, their cache causes the page to load faster on subsequent visits. Later work in the field abused remote timing attacks to disclose sensitive information about target users, such as the existence of active accounts [3] and the authentication state of a victim [25]. This line of work is important because it proved the practicality of remote timing attacks, which may sound far-fetched on paper; yet, prior work is based on simple empirical observations with limited scientific underpinnings. At a high level, prior work leverages the same intuitive idea: response times are collected and undergo some form of statistical analysis to determine whether they reveal any information about the target system. The key differences between different proposals lie in the attack's objectives, the details about data collection, and the specific nature of the statistical analysis, e.g., a simple visual inspection of histograms rather than the adoption of a statistical test like the t-test.

Inspired by the rich research line on queuing theory [2, 30], in this paper we put forward a novel approach to remote timing attacks based on stochastic modeling. Queuing theory is a popular branch of applied mathematics widely used for performance engineering tasks. It resorts to stochastic models to predict quantitative behaviors of systems such as their response times. At first glance, queuing models may seem to rely on strong assumptions for numerical tractability, but experience in engineering practice has demonstrated their remarkable robustness allowing practitioners to adopt it for many purposes [22]. A queuing model can predict the expected response time of a network request given its type, e.g., a payment request. Here, we propose a novel approach that combines Bayesian statistics with queuing modeling to address the inverse problem: determining the likely type of a network request based on its observed response time.

This approach offers a potential new foundation for remote timing attacks by outlining a general framework that may help distinguish different types of requests based on timing information. Notably, the relevance of this perspective is not purely theoretical; it also offers practical advantages. Our method appears naturally suited to handle the complexities that make performance evaluation challenging, such as the impact of the load factor on the target system's response times — factors that can sometimes limit the robustness of purely empirical attacks. Moreover, our attack establishes quantitative probability estimates about the predicted request type, thus providing a formal assurance about the estimated accuracy of the attack, and leverages the generality of our key idea to be uniformly applied to different settings. Even from a purely experimental perspective, our attack is superior to the state of the art. It outperforms the classic box test by Crosby et al. [7] in terms of number of successful predictions and the BakingTimer attack [25] in terms of all the considered performance measures. We believe that the formal foundations put forward in this paper can be successfully applied to the creation of principled remote timing attacks which are more effective in practice, because better equipped to deal with the complexity of the problem they are trying to solve.

Contributions

In this paper, we make the following contributions:

(1) We propose a new timing attack strategy that builds on the combined use of Bayesian statistics and queuing modeling to improve existing empirical approaches proposed in the

Proceedings on Privacy Enhancing Technologies 2025(3), 545–559

https://doi.org/10.56553/popets-2025-0112

literature (Section 3). By leveraging the quantitative analysis supported by stochastic models, our approach aims to address key limitations in current methods, offering a more robust framework for timing attacks (Section 4).

(2) We propose a specific instantiation of the stochastic model supporting our timing attack based on a M/M/1 Processor Sharing (M/M/1/PS) queuing system [11, 17]. We use standard mathematical tools to support quantitative analyses designed to implement effective remote timing attacks in practice (Section 5).

(3) We implement our new timing attack and we compare its effectiveness against existing state-of-the-art attacks, discussing the key advantages enabled by the adoption of our proposal (Section 6).

To support reproducibility and fully comply with open science policies, we make the code of our research and the collected data available for review [4].

2 Background

We here review selected technical ingredients required to appreciate the paper. We also introduce the key terminology and notation.

2.1 Remote Timing Attacks

In a remote timing attack, the attacker infers information about server-side computations based on measured response times, e.g., to determine state information about other users of the system [7]. More specifically, we assume that each computation belongs to one of $k \geq 2$ possible classes and that the attacker measures the response times of network requests to determine the class of the corresponding remote computation. For simplicity, we refer to the class of a request as the class of the associated server-side computation.

To make the explanation concrete, we focus on the web security setting and we assume that the attacker interacts with a remote web application using a web browser, but our approach is more general and works for any remote timing attack. Following the terminology by Bortz and Boneh [3], the attacker may perform two main classes of attacks:

- *Direct timing attacks*: the attacker sends arbitrary requests towards a target web application from their own browser and measures the response times to infer information about server-side secrets. For example, a web application supporting a login functionality may take more time to perform login attempts when the attacker provides a username associated with an existing account, as opposed to an invalid one. This way, the attacker may craft a list of target usernames to test and identify those corresponding to an existing account on the server by abusing the timing channel. Such attacks pose significant privacy risks, by allowing attackers to determine whether a specific individual has an account on sensitive websites, such as those related to adult content or political parties, information that users would likely wish to keep private. Furthermore, these attacks can enhance other malicious strategies, such as crafting personalized and more persuasive phishing attempts, thereby increasing the chances of successful exploitation.
- *Cross-site timing attacks*: the attacker sends arbitrary requests towards a target web application from the victim's browser, e.g., by serving JavaScript on a malicious web page, and measures the response times to infer information about the victim's state on the server. For example, a web application implementing a private area might take more time to generate a valid profile page than an error page denying access to the profile. This way, the attacker may send a request to the profile page from the victim's browser and determine the victim's authentication state on the server by abusing the timing channel. These attacks pose a serious threat to user privacy. For example, if an attacker targets a specialized medical center, such as one for HIV treatment or mental health services, determining that a user is logged in could expose highly sensitive health information, as it reveals their active and ongoing medical engagement. This exposure could lead to discrimination, blackmail, or significant distress if the information is misused.

Direct timing attacks are straightforward to attempt in practice, because everything is under the attacker's control. Cross-site timing attacks, instead, may be performed by any attacker with the (limited) capabilities of a traditional *web attacker* [1]. This attacker owns a malicious website hosting HTML and JavaScript, which is assumed to be visited by the victim. The attacker may target casual visitors or use social engineering techniques like phishing to lure selected victims into their website. Even though the victim uses a standard web browser that correctly enforces the Same Origin Policy (SOP), the attacker can measure timing information by means of JavaScript APIs such as the Performance API.

2.2 Terminology

This section introduces the terminology used throughout the paper, with a particular focus on concepts related to queuing theory. A *queuing system* is a mathematical abstraction of a facility providing a certain service to requests. Requests arrive at the system according to a random process, compete for access to resources and once they obtain the desired service, they leave the system. The scheduling discipline governs the assignment of the resources to the requests. In our model, we consider a system under the *Processor Sharing* (PS) scheduling policy [11, 17]. This scheduling policy is particularly important when representing real systems as it serves as the most straightforward abstraction of the often-used Round-robin approach. PS evenly distributes computational resources to pending requests. Hereafter, we introduce some important definitions.

Service time: this is the time that would be spent by a request at the system if there were no other competing requests. This is a random variable, since the system's internal behavior (e.g., caching policies, garbage collection, etc.) introduces some unpredictable delays that can be known only through their statistics. Coherently with the literature, we denote with μ the service rate, i.e., the average number of jobs served per second. Hence, the average service time μ^{-1} is the reciprocal of the service rate.

Response time: this is the time spent between the formulation of a request and its reply, without considering the network round-trip time. From the modeling perspective, this is a random variable denoted with the letter R.

Waiting time: this is the time a request spends in the system competing for the server's capacity with other requests. It excludes the actual service time the request would experience if it were the only one present. In a PS system, this time accounts for delays caused by simultaneous sharing of the server's resources with other active requests, and it is computed as the difference between the response and service times. In our queuing model, this is a random variable denoted with the letter W.

Arrival rate: The arrival rate is the expected number of requests arriving at the system per unit of time. We denote this quantity by λ.

Load factor: The load factor is the ratio between the arrival rate λ and the maximum service capacity of the system. This is denoted by ρ and is a pure number. For single server systems, $\rho = \lambda/\mu$ and $1 - \rho$ is the probability of the empty system for stable systems, i.e., systems where the arrival rate is less than the service rate ($\rho < 1$).

Stationary system: Intuitively, a stable system is considered stationary when ρ remains constant over a sufficiently long period. It is important to note that ρ refers to an average value, so requiring it to be constant does not mean that the resource occupancy is unchanging at every moment. Fluctuations can still occur (and in fact, they often do), but over time, the statistical properties of these fluctuations become stable. This assumption, while standard in queuing theory, is also crucial for applying most of its results. In fact, these results are typically derived under the premise that the system's *statistical* behavior does not change over time. Stationarity not only simplifies the mathematical analysis but also ensures that *key performance metrics*, such as waiting and response times, are well-defined and meaningful. Throughout the rest of this paper, we will assume that the systems under consideration are stationary.

3 Motivation and Contribution

Remote timing attacks leverage the observation that the response times of network requests may reveal information about server-side computations. Existing remote timing attacks are all based on the same high-level idea: response times are collected and undergo some form of statistical analysis to determine whether they reveal some information about the target. The innovative aspect of the approach proposed in this paper lies in its departure from simply collecting response times. Instead, it leverages stochastic models (such as queuing models and Bayesian models) in a novel way to enhance and complement the information gathered from response times. By doing so, it addresses the limitations of previous approaches that relied solely on basic response time collection. This approach provides a more robust framework for analyzing system performance by integrating stochastic modeling, thus improving on the inherent constraints of existing methods. Specifically, the proposed approach enhances the accuracy and practicality of remote timing attacks by accounting for the server's load factor, leading to a highly predictive model reducing the number of requests required to carry out a successful attack.

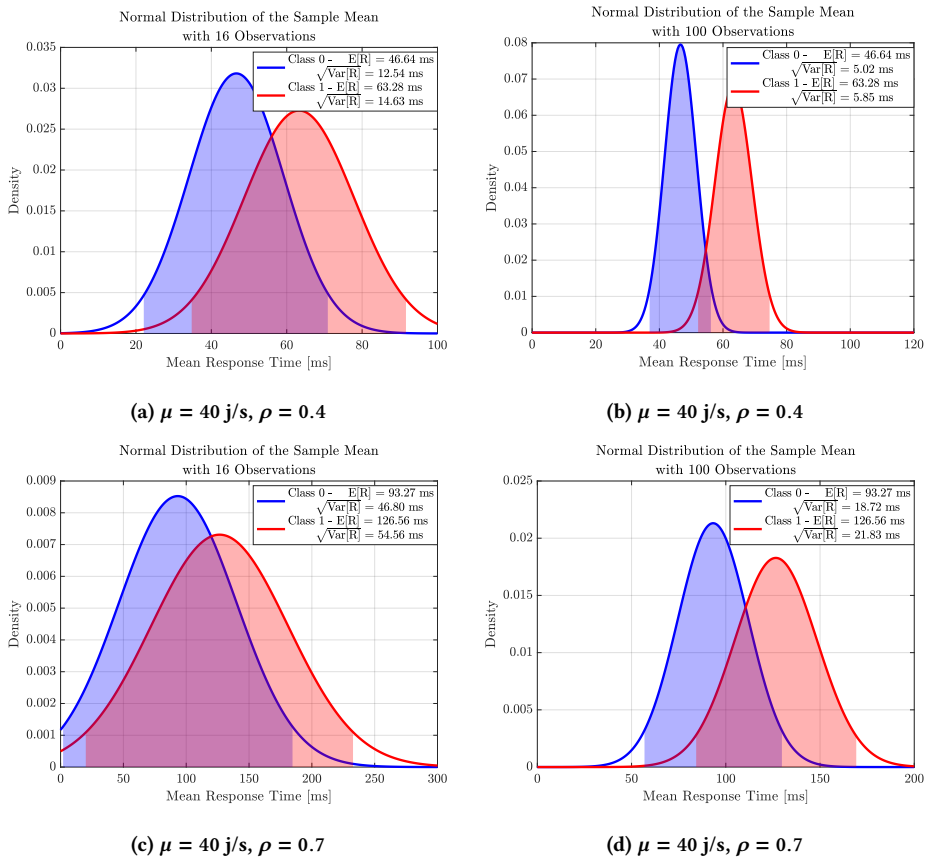
3.1 Response Time vs. Service Time

A key observation of our research is that timing attacks are difficult to carry out in practice because they abuse differences in the service time, but the attacker can only measure the response time. These two notions roughly coincide when the target web application is under a low load factor $\rho \approx 0$, but the response time may significantly differ from the service time when the load factor increases and gets closer to 1. To understand this from an intuitive perspective, assume that the attacker infers a time difference of 10 ms between authenticated and unauthenticated requests based on observations drawn at 9 AM. Later in the day, say at 5 PM when the target web application is under a higher load factor, the victim accesses the attacker's website and the attacker mounts a timing attack on the target web application. Since the response time of the web application is inflated, a difference of 10 ms may not be as statistically significant as the one observed for the lower load factor observed at 9 AM. This issue has only been marginally considered in prior work and just from an empirical perspective, e.g., the authors of the BakingTimer attack [25] present a small-scale experiment showing that the time of the day has limited impact on the effectiveness of their attack, but this informal and limited evaluation should be considered with some reservations.

To clarify that our criticism is not just a speculation, Figure 1 reports the mean response times computed for two classes of requests (authenticated vs. unauthenticated) of a local installation of the popular WordPress web application. Intuitively, the difference in the response times may be abused to carry out a cross-site timing attack breaching the victim's privacy by inferring their authentication state. The figure shows that the load factor ρ may have a significant impact on the precision of the attack. For $\rho = 0.4$, the two distributions already have a significant overlap when they are estimated by collecting the response times of 16 requests for each class (Figure 1a). Increasing the number of requests to 100 mitigates this issue and allows one to more effectively discriminate the two distributions (Figure 1b). Unfortunately, when the load factor increases to $\rho = 0.7$, discriminating the two distributions is virtually impossible by collecting the response times of 16 requests (Figure 1c) and still difficult when increasing the sampling to 100 requests (Figure 1d). Existing empirical attacks take the number of requests to perform as a parameter, but our experiment shows that this is impractical, because the attacker has no proper way to set this parameter a priori. The straw-man approach of performing as many requests as possible is sub-optimal, because sending too many requests may lead to rate limiting, thus making the attack ineffective. Moreover, the attacker may only have a limited time window to carry out their attack, e.g., because the victim navigates away from the attacker's page.

3.2 Proposed Approach

Figure 2 sketches the attack methodology proposed in our paper. Rather than using the measured response times directly to draw conclusions, the attacker uses them to construct a queuing model



(a) $\mu = 40$ j/s, $\rho = 0.4$ **(b) $\mu = 40$ j/s, $\rho = 0.4$**

(c) $\mu = 40$ j/s, $\rho = 0.7$ **(d) $\mu = 40$ j/s, $\rho = 0.7$**

Figure 1: Response time distributions under different load factors ρ for a fixed service rate μ.

of the target web application. Traditionally, queuing systems take the arrival rate λ and service rate μ as *input* parameters to compute key performance metrics such as the system's average response and waiting times. By contrast, with our approach we reverse this process: by collecting response time data from a real system, we leverage queuing theory to estimate these parameters, and construct an abstraction of the real system. The queuing model obtained from the estimated parameters determines the probability distribution of service times for requests in different classes. This information can then be given as input to a Bayesian statistical model that addresses the question: given a set of observed times, what is the probability that a request belongs to a particular class?

This approach has many distinctive advantages. Besides its built-in support for the challenges posed by the load factor, it uses quantitative probability estimates to determine when enough requests have been sent to be confident about the outcome of the attack, hence no a priori bound on the number of requests to send is required. Our experimental evaluation in Section 6 shows that the number of requests required to achieve strong probabilistic guarantees is small and significantly lower than for state-of-the-art attacks. Moreover, our attack is general, because it uniformly supports both direct timing attacks and cross-site timing attacks, while making few assumptions about the attacker's capabilities. Our attack can target any scenario where server-side computations can be classified in $k \geq 2$ classes based on their service times. We require that the attacker can use their own browser to send HTTP requests belonging to the k classes, e.g., authenticated requests including cookies and unauthenticated requests without cookies. However, the attacker does not know in advance whether these requests are useful to discriminate the classes, but rather uses the stochastic model constructed from the response times to identify potential room for exploitation. After the model is constructed, the attacker may determine the class associated with a given set of requests based on their measured response times alone.

4 Attack Description

Our attack operates in two phases, called *exploration* and *exploitation*, respectively. During the exploration phase, the attacker collects timing information about the target web application at different times and for different classes of requests. The exploitation phase leverages the information collected in the exploration phase to build a stochastic model of the target web application and actually mount the attack. In particular, the attacker measures the response times of requests of an unknown class and uses the model to reconstruct their class. While we describe the attack assuming that the

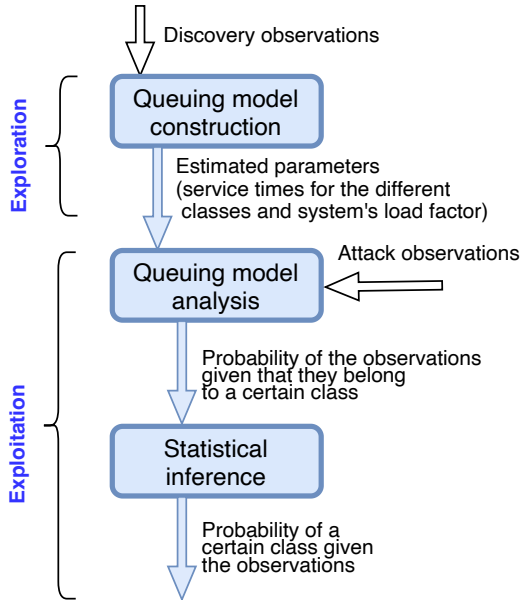


Figure 2: Sketch of the methodology proposed in this paper.

attacker wants to discriminate two possible classes for simplicity, the extension to $k > 2$ classes is straightforward.

4.1 Preliminaries

We present the two phases of our attack using pseudo-code. Both the exploration phase and the exploitation phase need to measure response times, which are then fed to our stochastic model. However, when we measure the time between an HTTP request and its response, we are also measuring the round-trip time (RTT). Since our model does not account for RTT or its variability, we need to eliminate its impact from our measurements. We achieve this by empirically measuring it and subtracting it from our data. To obtain a reliable estimate of response times, we employ two sub-procedures.

The first sub-procedure TimeRequest(u) sends an HTTP request to the URL u and estimates its response time. This sub-procedure is used in the exploitation phase and is implemented as follows. We first send an HTTP request r to u and we measure the time t_r until its response. t_r includes both the response time of r and RTT. Along with request r, we craft another one r', whose time until response t'_r serves as a possible estimate of RTT. An example of requests that are a suitable choice for this purpose are those triggering a redirect. The reason behind this is that redirects often require negligible server-side computation. One effective way to craft a redirect request is by modifying the protocol of URL u from HTTPS to HTTP. If the target site does not enforce HTTP Strict Transport Security (HSTS), this HTTP request is likely to trigger a redirect to the HTTPS version of the site. We can then estimate the response time of r by subtracting the estimated RTT from t_r.

The second sub-procedure TimeRequestWithClass(u, i) sends an HTTP request of class i to the URL u and estimates its response time. This sub-procedure is used in the exploration phase, hence it assumes that the attacker is able to craft a request of class i. The way the attacker does that depends on the specific class of requests. For example, authenticated requests require the attacker to be able to login to the target web application, while unauthenticated requests can always be forged. The estimate of the response time is performed using the same technique proposed for the TimeRequest(u) sub-procedure, just the format of the request r will be different depending on its class.

Finally, our pseudo-code relies on the construction of a stochastic model M, which is queried to determine probability estimates of the class of a target request. Model construction and interrogation are implemented by the sub-procedures BuildModel and EstimateProba, which are discussed in Section 5. In this section, we consider the model M as a black-box. Additional implementation details of our attack are described in Appendix A.

Algorithm 1 Exploration Phase

```
1:  function Explore(u, endtime, tick, samples)
2:      R_0 ← {}                    ▷ Response times for class 0 (empty dict.)
3:      R_1 ← {}                    ▷ Response times for class 1 (empty dict.)
4:      b_0 ← +∞                    ▷ Service time estimate for class 0
5:      b_1 ← +∞                    ▷ Service time estimate for class 1
6:      t_0 ← 0
7:      t_1 ← 0
8:      t ← GetCurrentTime()        ▷ Start time of exploration
9:      while t ≤ endtime do
10:         R_0[t] ← []             ▷ R_0[t] is initialized to the empty list
11:         R_1[t] ← []             ▷ R_1[t] is initialized to the empty list
12:         reqs ← 0
13:         while reqs < samples do
14:             t_r ← TimeRequestWithClass(u, 0)
15:             R_0[t] ← t_r :: R_0[t]   ▷ Append t_r to the list R_0[t]
16:             t_r ← TimeRequestWithClass(u, 1)
17:             R_1[t] ← t_r :: R_1[t]   ▷ Append t_1 to the list R_1[t]
18:             reqs ← reqs + 1
19:         if Mean(R_0[t]) < b_0 then
20:             b_0 ← Mean(R_0[t])
21:             t_0 ← t
22:         if Mean(R_1[t]) < b_1 then
23:             b_1 ← Mean(R_1[t])
24:             t_1 ← t
25:         t ← t + tick
26:         WaitUntil(t)            ▷ Wait until the next tick
27:     return (R_0[t_0], R_1[t_1], R_0[t])
```

4.2 Exploration Phase

In the exploration phase, the attacker collects timing information about the target web application, which is used to construct the stochastic model underlying the exploitation phase. The exploration phase is done from the attacker's browser. We present exploration before exploitation because model construction is a precondition to exploitation, however the attacker can extend the exploration phase to run it concurrently with the exploitation phase. This way, the attacker constantly collects up-to-date timing information leading to the creation of more accurate models for the exploitation phase.

Algorithm 2 Exploitation Phase

```
1:  function Exploit(u, X^0, X^1, X^R, α, samples)
2:      class ← ⊥                          ▷ The request class is initially unknown
3:      X^A ← []                           ▷ Response times used to mount the attack
4:      reqs ← 0
5:      M ← BuildModel(X^0, X^1, X^R)
6:      while class = ⊥ ∧ reqs < samples do
7:          t_r ← TimeRequest(u)
8:          X^A ← t_r :: X^A               ▷ Append t_r to the list X^A
9:          (p_0, p_1) ← EstimateProba(M, X^A)
10:         if p_0 ≥ α ∨ p_1 ≥ α then
11:             class ← arg max_{i∈{0,1}} (p_i)
12:         reqs ← reqs + 1
13:     return class
```

The Explore procedure shown in Algorithm 1 takes as input the target URL u, the end time of the exploration phase $endtime$, a time delta $tick$ defining the frequency of the data collection, and the number of response time estimates $samples$ to collect for each class at each tick. The procedure uses dictionaries R_0, R_1 associating to each time tick t a list of response times for the two request classes. It computes b_0, b_1, for class 0 and class 1 respectively, by looking for the time ticks t_0, t_1 where the mean of the response times is smallest, i.e., the load factor of the target system is the lowest observable. The procedure returns three lists of timing observations: $R_0[t_0]$ is a list of response times for class 0 under the lowest observed load factor, $R_1[t_1]$ is a list of response times for class 1 under the lowest observed load factor, and $R_0[t]$ contains the most recently measured response times for the class 0. The three lists are subsequently passed as inputs X^0, X^1 and X^R of the Exploit procedure, presented in the next section.

If the exploration phase starts at time $starttime$, the total number of requests sent to the target URL is $2 \cdot \lfloor (endtime - starttime)/tick \rfloor \cdot samples$. Data collection is performed periodically to be able to reconstruct accurate estimates of the load factor at different times, leading to the construction of constantly updated models.

4.3 Exploitation Phase

In the exploitation phase, the attacker uses the information collected in the exploration phase to build a stochastic model used to infer the class of the target requests. In direct timing attacks, the exploitation is carried out from the attacker's browser. Conversely, in cross-site timing attacks, the exploitation occurs from the victim's browser.

The Exploit procedure shown in Algorithm 2 takes as input the target URL u, the lists of timing observations X^0, X^1, X^R coming from the exploration phase, the confidence $\alpha \in (0, 1]$ and the maximum number of requests $samples$ that the attack can perform. The attack starts by using parameters X^0, X^1, and X^R to build the stochastic model M used to perform the attack. Intuitively, X^0, X^1 are used to estimate the service times of the two classes, while X^R is used to estimate the most recent load factor.

The attack is implemented by means of a loop, which stops when either the class of the target requests has been disclosed or the maximum number of requests has been sent. Each iteration of the loop estimates the response time at u and uses the stochastic model M to determine the probabilities (p_0, p_1) of the two classes based on the collected response times X^A. If one of these probabilities reaches the confidence α, the corresponding class is returned. When the desired confidence is not reached within the maximum number of iterations, the unknown class $\perp$ is returned.

5 Stochastic Models

We finally present the formal models used in our attack. We first introduce the queuing model used to represent the target web application. Since this model needs to be initialized with appropriate parameters to predict service times, we then discuss how we estimate such parameters from black-box observations of the target web application. Finally, we present the statistical model that leverages the queuing model to answer its inverse question, i.e., what is the likely class of a request given an estimate of its service time. These two models are the ones used by the BuildModel and EstimateProba sub-procedures respectively (Section 4).

5.1 Queuing Model

Let us model the system under consideration with a M/M/1/PS queuing model [11, 17]. This means we assume that the system is composed of one single server, and it is subject to a Poisson arrival process, i.e., in an interval of Δt seconds, the number of requests that arrive at the system is distributed according to an independent Poisson random variable with mean $\lambda \Delta t$, where λ (measured in requests per second) is the arrival rate. Jobs are served according to a Processor Sharing (PS) policy, with service time distributed according to a negative-exponential random variable with mean μ^{-1}, so that μ is the jobs' service rate. PS is a service policy often studied in queuing theory where all jobs are served concurrently, with each one receiving an equal share of the available service capacity, which is a realistic assumption for modern web servers. In this system, every job begins receiving service immediately, thereby eliminating the time spent waiting in a queue. So, in the setting we are describing, the waiting time for each job is considered to be the difference between each job's effective service time and its response time. Both the assumption of the system being subject to a Poisson arrival process and the service time being distributed according to a negative-exponential random variable are standard assumptions that allow tractability and have been shown to be robust [20, 27].

Although real systems are not usually single servers, M/M/1/PS can be successfully used to understand their quantitative behavior, assuming that the single server abstraction accounts for the entire service capacity of the real system [6, 27]. While this simplification remains effective also for distributed systems as confirmed in this paper, the actual number of servers behind a web application is often unknown, and requests may be routed to different servers with varying workloads. This means the model may not fully capture the exact workloads across all servers involved, which, in some cases, could result in a loss of prediction accuracy.

This specific queuing system has been extensively studied in the literature, and as a result, many key performance metrics have well-known analytical expressions, under the assumption that the system is stationary. This assumption is not only standard in the queuing theory literature but also reasonable in this particular setup, because load variations tend to fluctuate over time but do

so quite slowly with respect to service times [29]. The equations we are about to present give the core quantities required for our methodology. Eq. (1)-(3) are essential for the procedure detailed in Section 5.2, which estimates the system parameters λ and μ. In general, considering a specific job with a deterministic service time τ, the expectation of the response time is given by:

$$E[R|\tau] = \frac{\tau}{1-\rho}\,. \tag{1}$$

The probability of finding the system empty is computed as:

$$\pi_0 = 1-\rho\,, \tag{2}$$

thus, the probability for an incoming job to have waiting time equal to 0 can be expressed as:

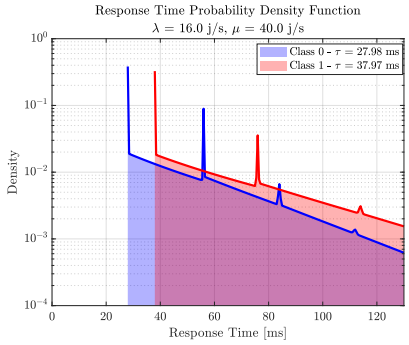
$$\Pr\{W=0\} = \pi_0 e^{-\lambda\tau}\,, \tag{3}$$

that represents the probability of both finding the system empty and ensuring that no other job arrives before the specific observed job is completed. Additionally, the literature provides also the Laplace-Stieltjes transform of the waiting time distribution for an arriving job in a M/M/1/PS system, conditioned on the deterministic service time required by that specific job. This expression was first given in [15, Eq. (30)] and it is further detailed in Appendix B, while here we simply denote it as:

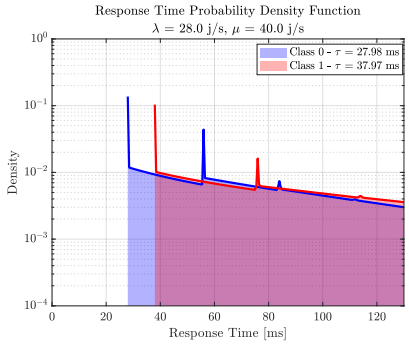
$$F_W(x|\tau) = Pr\{\text{waiting time} \le x | \text{service time} = \tau\}\,. \tag{4}$$

$F_W(x|\tau)$ is used in Section 5.3 to estimate the probability density of an observation of the waiting time x given an instance of the service time τ. As reported in the Appendix, this expression depends on the system parameters λ and μ and can be numerically evaluated for each x starting from the expression of [15, Eq. (30)].

Remark. Figure 3 illustrates the response time distribution of a M/M/1/PS conditioned to the deterministic service time of a job. This corresponds to the scenario in which an observer can measure the response time of a job whose service time is known, but the other incoming traffic maintains its stochastic nature. The observed task belongs to one of the two classes of requests under study (identified as 0 and 1), computed under different load factors. The deterministic service times distinguishing these classes are derived from empirical observations. The solid lines represent the probability density function (pdf) of the response times computed using the results derived in [15]. In practice, when the plots, for a certain response time, show similar densities (e.g., the values between 85ms and 110ms in Figure 3b), it is very hard to distinguish between the two different classes of service. However, notice that in both cases there exists an area between the service time of class 0 and that of class 1 where it is possible to distinguish between the two service classes clearly. However, the probability of this area becomes small as the load factor increases. Coherently with the observations proposed for Figure 1, it is shown that the distinction of the two classes becomes more challenging as the load factor increases. This trend is evident when comparing Figures 3a and 3b, which correspond to load factors $\rho = 0.4$ and $\rho = 0.7$, respectively. As shown, at higher load factors, overlapping regions between the classes are more prevalent.



(a) $\lambda = 16$ j/s, $\mu = 40$ j/s, $\rho = 0.4$



(b) $\lambda = 28$ j/s, $\mu = 40$ j/s, $\rho = 0.7$

Figure 3: Response time distributions for deterministic service times.

It is worth noticing that the behavior of the pdf of the response time has some spikes. These are placed at the occurrences of the integer multiples n of the service times and correspond to the events in which the observed jobs arrive at the system with n jobs in service, without any other arrivals or departures during their service. These observations are important to understand the Bayesian model presented in Section 5.3 where, differently from here, we consider the possibility of noisy measurements as it usually happens in real-world scenarios.

5.2 Parameter Estimation

In this section, we present the methodology that we use to estimate the parameters needed for the model. In particular, we estimate parameters λ, μ, and the two classes of requests service times by recording the following collections of measurements:

- $X^0 = (X_1^0, X_2^0, \ldots, X_{N^0}^0)$: response times measured for requests of class 0 under the lowest observed load factor.
- $X^1 = (X_1^1, X_2^1, \ldots, X_{N^1}^1)$: response times measured for requests of class 1 under the lowest observed load factor.
- $X^R = (X_1^R, X_2^R, \ldots, X_{N^R}^R)$: response times for requests of class 0 under the most recent load factor.

Since the system appears as a black-box to the attacker, it is not possible to precisely determine the system's load factor at any given moment or other quantitative characteristics. Consequently,

parameter estimation relies on a series of reasonable assumptions and leverages properties of the collected observations. As discussed in Section 4.2, the vectors X^0 and X^1 are inferred from a subset of observations with the lowest averages. These observations are selected to represent conditions of zero waiting time, providing the most accurate estimates of the service times for the classes under minimal load conditions. Similarly, the vector X^R is taken from the most recent set of observations, capturing the system's latest operational state. This approach facilitates the estimation of service times for both classes and the system's load factor using the methodology outlined below.

Estimation of the service time distributions. The first step in our procedure consists of estimating the distribution of the service times for class ℓ, where $\ell = 0, 1$.

- We start by grouping observations X^ℓ into bins, each represented by a centroid and an associated probability. This approach allows us to account for the variability inherent in real service times. We define:

$$X^\ell_{\min} = \min_{1\le i\le N^\ell}\{X^\ell_i\} \quad \text{and} \quad X^\ell_{\max} = \max_{1\le i\le N^\ell}\{X^\ell_i\}.$$

- We partition the range $X^\ell_{\max} - X^\ell_{\min}$ into K contiguous and non-overlapping subranges, namely *bins*. We denote them as B^ℓ_j, $j = 1, 2, \ldots, K$. We group the X^ℓ observations according to the ranges of values associated with each bin and denote with N^ℓ_j the number of observations that belong to the j−th bin. The empirical probability of each bin is then defined as:

$$p^\ell_j = \frac{N^\ell_j}{N^\ell}, \quad j = 1, \ldots, K.$$

We denote with C^ℓ_j, the *centroid* of B^ℓ_j, $j = 1, \ldots, K$, i.e., the average of observations belonging to B^ℓ_j.

Figure 4: Discretization of the empirical service time distribution for a particular instance of class 0 and 1.

In this way, we obtain a discretization of the empirical service time distribution for both classes as illustrated in Figure 4. Notice that this phase must be done only once.

Estimation of $\widehat{\rho}$, $\widehat{\lambda}$, $\widehat{\mu}$. In this phase, we estimate the model parameters that depend on the work that is being performed by the system in a certain moment, namely, the load factor as well as the arrival and service rates. We need to consider just one of the two classes, e.g., class 0.

- The system's load factor, given $X^R_1 = x^R_1, \ldots, X^R_{N^0} = x^R_{N^0}$, is:

$$\widehat{\rho} = 1 - \frac{\sum_{j=1}^K C^0_j\, p^0_j}{\left(\sum_{j=1}^{N^R} x^R_j\right)/N^R}. \tag{5}$$

This expression can be derived from Eq. (1). In fact, the expression $\sum_{j=1}^{N^R} x^R_j/N^R$ is the estimate of the average response time $\widehat{E[R]}$, thus we have, by the law of total probability applied to Eq. (1):

$$\widehat{E[R]} = \frac{\sum_{j=1}^K C^0_j p^0_j}{1-\widehat{\rho}}.$$

It is now sufficient to explicit $\widehat{\rho}$ to obtain Eq. (5).

- We now estimate the probability for a user to observe zero (or very low) waiting time. Since we do not know the exact value of the service time for the requests belonging to class 0, but only an estimate characterized by an empirical distribution given by the above bins, we can first compute an estimate of $\widehat{\Pr}_j\{W = 0\}$, $j = 1, \ldots, K$ by successively conditioning on the reference value of the j-th centroid as service time. This is done by computing the proportion of X^R observations falling inside the interval $[C^0_j(1-\delta), C^0_j(1+\delta)], \forall j = 1, \ldots, K$, and where δ denotes a small, arbitrarily chosen threshold.
- Having computed $\widehat{\rho}$ and $\widehat{\Pr}_j\{W = 0\}$, we are now able to invert Eq. (3) and obtain a value $\widehat{\lambda}_j$ for each specific $\tau_j (= C^0_j)$ by inverting Eq. (3), explicitly:

$$\widehat{\lambda}_j = -\frac{1}{C^0_j}\ln\left(\frac{\widehat{\Pr}_j\{W=0\}}{1-\widehat{\rho}}\right), \quad j = 1, \ldots, K.$$

So to obtain $\widehat{\lambda}$ as follows:

$$\widehat{\lambda} = \sum_{i=1}^K p^0_i \widehat{\lambda}_i.$$

Once we have done this last step, it is straightforward to compute $\widehat{\mu}$ from the load factor relationship given by $\rho = \lambda/\mu$. The methodology described in this section is included in Algorithm 2 through the BuildModel sub-procedure.

5.3 Statistical Model

In this section, we present the statistical model designed to estimate the posterior probabilities that a user belongs to a specific class, based on observations of response times and system parameters estimated in the previous section. This model will be used by the EstimateProba sub-procedure (Algorithm 2) which implements the evaluation of Eq. (9). Hereafter, we show the mathematical derivation of the result.

The model uses the estimated parameters $\widehat{\lambda}$ (arrival rate) and $\widehat{\mu}$ (service rate) to account for the load factor of the underlying queuing model. The queuing model enables us to derive the response time conditioned on the observed job service time. We adopt a

Bayesian framework where the parameter Θ is a random variable restricted to the discrete space $\Omega = \{\tau_0, \tau_1\}$. Recall that the response time R consists of the sum of the service time, determined by the specific instance of the parameter Θ, and the waiting time W.

$$F_R(x|\Theta) = \begin{cases} F_W(x-\Theta|\Theta) & \text{if } x-\Theta > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Hereafter, we consider that the observations of the response times do not include the network round-trip times. We define the probability density function of an observation x as:

$$f_X(x|\Theta) = \frac{F_R((x+\epsilon)|\Theta) - F_R((x-\epsilon)|\Theta))}{2\epsilon}, \tag{6}$$

for some positive small ϵ. Basically, Eq. (6) is a smoothed pdf with respect to $\partial F_R(x|\Theta)/\partial x$ that does not change the mean of the random variable R given Θ. The importance of the smoothening is evident from the plots reported in Figure 3, where we can see that, especially for low load factors, the response time pdf concentrates heavy masses of probability around some spikes. Using the difference between the values of the cumulative distribution functions makes the evaluation of $F_R(\cdot|\Theta)$ more robust to measurement errors. The choice of an appropriate value of ϵ depends on the measurement settings, such as the variability of the measured round-trip time.

During our attack, we collect a set of observations X^A (recall that these vectors do not contain the round-trip times), and the aim of the Bayesian estimation is to infer the probability $\Theta = \tau_1$ or, equivalently, its complement. The likelihood function of the sequence is:

$$f_{X^A}(x_1^A, \dots x_{N^A}^A|\Theta) = \prod_{i=1}^{n} f_X(x_i^A|\Theta), \tag{7}$$

where the x_i^A are the actual values of the collection of observations that we denoted with X^A. Let $Pr\{\Theta = \tau_1\} = P_{\tau_1}$ be the prior probability of τ_1, and $Pr\{\Theta = \tau_0\} = 1 - Pr\{\Theta = \tau_1\} = P_{\tau_0}$ the prior probability of τ_0.[1]

According to Bayesian statistics, the probability $\Theta = \tau_1$ ($\Theta = \tau_0$), given the observations, i.e., the posterior probability, is:

$$g_{X^A}(\theta|x_1^A, \dots, x_{N^A}^A) = \frac{f_X(x_1^A, \dots, x_{N^A}^A|\theta)P_\theta}{G}, \tag{8}$$

where $\theta \in \{\tau_1, \tau_0\}$ and G is a probability normalizing constant defined as:

$$G = f_X(x_1^A, \dots, x_{N^A}^A|\tau_1)P_{\tau_1} + f_X(x_1^A, \dots, x_{N^A}^A|\tau_0)P_{\tau_0}.$$

The following theorem shows that the method that we propose is sound and that, with a sufficient number of observations, converges almost surely (i.e., with probability 1) to the correct answer. It uses the notion of *consistent estimator*. Intuitively, an estimator is consistent if, as the number of samples grows, it becomes more and more accurate. Informally, Theorem 5.1 states that if it is possible to infer the correct value of Θ by collecting the observations of the response times, then our procedure will converge there, i.e., the posterior distribution will concentrate around the correct parameter value, as more data is observed.

Theorem 5.1. *For sufficiently small ϵ, formally $\epsilon \to 0$, given the indicator function:*

$$I_A(\Theta) = \begin{cases} 1 & \textit{if } \Theta = A \\ 0 & \textit{otherwise} \end{cases}.$$

If there exists a consistent estimator of the parameter Θ, then the posterior probability given by Eq. (8) converges almost surely, i.e., with probability 1, as $n \to \infty$, to $I_A(\Theta)$, where $A = \tau_1$ or $A = \tau_0$.

The result is an immediate application of Doob's theorem (see, e.g., [26, Th. 7.78]). Notice that Theorem 5.1 requires the existence of a consistent estimator, hence we provide the following theorem:

Theorem 5.2. *For sufficiently small values of ϵ, formally ($\epsilon \to 0$), there exists a consistent estimator for Θ.*

Proof. It is sufficient to notice that under the assumption $\epsilon \to 0$, $f_X(\cdot|\Theta)$ as defined by Eq. (6) is the probability density function of the random variable modeling the response time of the system. As a consequence, there exists the consistent estimator given by the Maximum Likelihood method as proved in [26, Th. 7.48]. □

The statements of Theorems 5.1 and 5.2 hold under milder but more technical assumptions. It is sufficient that ϵ does not change the pdf of Eq. (6) with respect to the correct one $f_R(x|\Theta)$ in such a way that we cannot distinguish $f_X(x|\tau_0)$ from $f_X(x|\tau_1)$ anymore.

For a given number of attack observations, N^A, we apply Eq. (8) to compute the probability that the parameter Θ is τ_1 (τ_0). More specifically, we iterate over all possible combinations of the previously computed centroids for the service time distributions, C_i^0, C_j^1, where $i, j = 1, \dots, K$, weighting the results by the product of the probability associated with the two centroids, that is:

$$Pr\{\Theta = \tau_1|x_1^A, \dots, x_n^A\} \simeq \sum_{j=1}^{K}\sum_{i=1}^{K} p_j^0 p_i^1 g_{X^A}(C_i^1|x_1^A, \dots, x_N^A). \tag{9}$$

The denominator of Eq. (8) is computed at each iteration using C_i^0 and C_j^1 for τ_0 and τ_1, respectively. Further implementation details can be found in Appendix C.

6 Experimental Evaluation

In this section, we demonstrate our attack and compare its effectiveness against existing baselines. Research ethics are discussed in Appendix D.

6.1 Experimental Setup

We test the effectiveness of our attack in two different settings. The first setting is *controlled*, i.e., we target local installations of two popular web applications (HotCRP and WordPress). The use of a controlled environment has several benefits. First, we have full control over the system's load factor. Since a key motivation of our research is that the load factor may negatively affect the accuracy of remote timing attacks, testing different load factors is useful to confirm this hypothesis. Moreover, the use of local installations allows us to factor out network delays and other sources of non-determinism introduced by network communication. This way, we

[1] Notice that, without any prior knowledge of the system $P_{\tau_0} = P_{\tau_1} = 1/2$. However, in different scenarios, it may be useful to set these parameters differently. For example, if it is known that only 10% of users are expected to belong to the class whose service time is τ_1, then the model can encompass this knowledge in its estimations. In our experiments, we always assume the most challenging scenario for the model, i.e., 1/2 for both.

can assess the quality of our model under ideal conditions, which is important to confirm that the choice of our formalism is well-grounded. In the second part of our experimental evaluation, we move away from our controlled environment and we test our attack against 20 popular websites of the Tranco ranking [24]. By doing this, we confirm that our attack can successfully target unknown web applications served over the Internet.

Our evaluation considers both direct timing attacks and cross-site timing attacks, where the attacker's goal is to disclose the class of an unknown request. We consider just two request classes, that we refer to as positive and negative. We compare the effectiveness of our attack against appropriate baselines from the literature using standard performance measures.

6.1.1 Baselines. Existing remote timing attacks are often informally presented and they are difficult to reproduce. Starting from the description in the original papers [7, 25], we re-implemented two existing attacks as specific instantiations of a general attack strategy, where the attacker collects some sets of timing observations and uses statistical tests to infer the request class. More concretely, the attacker first collects sets of timing observations Y_1, Y_2 for the two classes to discriminate. If a statistical test determines that Y_1 and Y_2 likely come from different distributions, the attack can take place. The attacker then collects a set of timing observations Z of some unknown class, which they want to disclose. Finally, Z undergoes a statistical test against both Y_1 and Y_2. If exactly one of the two tests succeeds, the attacker performs the prediction recommended by the successful test, otherwise there is not enough statistical evidence to predict any class (abstention).

In the case of direct timing attacks, we consider an attack based on the classic box test proposed by Crosby et al. in their seminal analysis of remote timing attacks [7]. This test was introduced in a formal analysis designed to better understand previous attacks showing that it was possible to exfiltrate RSA private keys based on measured response times [5], hence it is a representative example of a direct timing attack. The goal of the box test is to determine whether two sets of response times Z and Y come from the same distribution or not. The box test is parametrized by two quantiles i, j, normally chosen from the low part of the distribution of the response times. To perform the box test, Z and Y are sorted and two intervals are formed from their quantiles: $[q_i(Z), q_j(Z)]$ and $[q_i(Y), q_j(Y)]$. The box test considers Z and Y to come from different distributions if and only if the two intervals do not overlap.

As for cross-site timing attacks, we compare the effectiveness of our attack against BakingTimer, a state-of-the-art web timing attack [25]. In the case of BakingTimer, the set Z contains "time deltas" estimating the server-side computation time in terms of the differences between a set of response times and a set of estimates of the network round-trip time, similar to what we do in our attack. Computing the set of time deltas Z requires sending $2 \cdot |Z|$ requests. BakingTimer uses the classic t-test to disclose the class of network requests in a cross-site setting, based on the time deltas Y_1, Y_2 of the two target classes.

6.1.2 Hyper-Parameters. Table 1 reports the hyper-parameters of the different attack strategies considered in our evaluation. We motivate the choice of the hyper-parameters as follows.

Table 1: Hyper-parameters of the experimental evaluation.

Attack	Parameter	Value
Box Test [7]	Quantiles	3%-5%
	Number of requests	50
Our Attack (Direct timing)	Probability threshold	90%
	Max. number of requests	50
BakingTimer [25]	Significance level	0.05
	Number of requests	40
Our Attack (Cross-site timing)	Probability threshold	90%
	Max. number of requests	40

For the box test attack, we set the two quantiles to 3% and 5% respectively, because the original paper recommends using values below 6% [7]. The number of requests is set to 50, so that we have enough observations to compute meaningful percentiles. For BakingTimer, we set the significance level for the t-test at 0.05, as it is a standard recommended practice, and we compare the resulting p-value to this threshold to determine statistical significance. The number of requests is set to 20, because the original paper reports satisfactory performance for any number between 10 and 50 [25]. We chose 20 requests as it represents a compromise between attack effectiveness and realism, considering the victim may remain on the attacker's page just for a short duration. Note that the use of 20 requests in the attack actually requires sending 40 requests to compute the time deltas, as reported in the table. Finally, for our attack we set the probability threshold α to 0.90, i.e., we require that the predicted class is assigned a probability of at least 90%. The maximum number of HTTP requests to send is set to the same number as its competitor to enable a fair comparison.

6.1.3 Measures. We compare the performance of different attacks using standard measures. We consider the *true positive rate* (TPR), i.e., the number of correct predictions out of all the attempted predictions with positive class, and the *true negative rate* (TNR), i.e., the number of correct predictions out of all the attempted predictions with negative class. Finally, the *abstention rate* (AR) measures the number of cases where the attacker does not predict any class out of all the possible attack attempts, due to insufficient statistical evidence about the correct class. We additionally compare attacks in terms of the number of sent requests, which is an important requirement for their practicality. Given that this is a difficult aspect to evaluate in a fair manner, we defer discussion to Section 6.4.

6.2 Experiments in a Controlled Setting

We first describe a set of experiments performed on local installations of HotCRP and WordPress. We experiment with direct timing attacks by mounting the account detection attack by Bortz and Boneh [3]. In direct timing attacks, both exploration and exploitation are performed from the attacker's browser. To conduct experiments in a controlled setting, the web applications were run on a server connected to the same wired network as the attacker's computer. For each web application, we register a new account and we forge an invalid username that does not correspond to any

Table 2: Comparison of the box test and our attack on key performance measures.

	Measure	Box Test	Our Attack
HotCRP	TPR	1.00	1.00
	TNR	1.00	1.00
	AR	0.39	0.00
WordPress	TPR	1.00	1.00
	TNR	1.00	1.00
	AR	0.44	0.00

registered user of the system. The attacker's goal is timing failed login attempts to determine whether a target username is associated with an account registered on the web application, so as to breach privacy. We compare the effectiveness of our attack against the box test attack, a classic representative of the family of direct timing attacks. We estimate the false positives and false negatives of the two attacks by conducting twenty attack attempts: ten with the existing account and ten with the invalid username. We replicate the experiments for four different values of the load factor obtained by increasing the number of users through the Grafana k6 load tester [18], leading to 80 attack attempts in total.

Table 2 reports the average results of our experiments. The results are similar for the two considered web applications. Both the box test and our attack are very accurate, because their TPR and TNR are 1 for all the different numbers of users, i.e., all the performed predictions are correct, irrespective of the load factor. However, there is a significant gap in the AR of the two methods: our attack is always able to make a correct prediction, while the box test cannot reliably make any prediction in approximately $40\% - 44\%$ of the cases. After checking the breakdown of the abstention rates for different numbers of users, we observe that the box test is heavily affected by the load of the system, as its abstention rate significantly fluctuates with changes in load. For example, in the case of HotCRP, we observe an abstention rate of approximately 20% and 50% under the lowest and highest loads respectively. In contrast, our attack is robust to variations in the load factor and is always able to perform reliable predictions.

6.3 Experiments in the Wild

We now move away from our controlled setting and we test the effectiveness of our attack against real-world websites. We experiment with cross-site timing attacks by mounting the login detection attack by Bortz and Boneh [3]. In cross-site timing attacks, the exploration is carried out from the attacker's browser, while the exploitation occurs from the victim's browser. To conduct these experiments we used two separate computers, one for the victim and one for the attacker, both connected to the same wired network and sending requests to the target website over the Internet. As a result, packets likely traveled through different paths and networks in this setup. The attacker's goal is to determine whether a target user is logged in on the web application or not, by collecting response times. We compare the effectiveness of our attack against BakingTimer [25], which is the latest variant of such cross-site login detection attacks. We estimate the false positives and false negatives of each attack by conducting sixteen attack attempts: eight in the logged-in state and eight in the logged-out state. We focus our investigation on the top 20 websites of the Tranco ranking [24] providing access to a private area and we assess their vulnerability to cross-site timing attacks. We manually navigate websites in decreasing order of popularity until we identify a login form, discarding websites where a login form is not identified within a reasonable amount of time or where registration cannot be readily performed, e.g., the website requires a paid subscription. For each website, we register a new account (if we do not already have one) and we perform login. After accessing the private area, we delete from the browser all the cookies whose SameSite attribute is not set to None and we manually check whether we are still logged in to the website. Websites which fail this manual test are not vulnerable to cross-site timing attacks, because their sessions cannot be established through cross-site requests [16]. Table 8 (Appendix E) reports the full list of websites where we successfully registered an account and logged in, including the potentially vulnerable ones. Perhaps surprisingly, 13 out of 20 analyzed websites establish sessions using cookies with the SameSite attribute deactivated.

We perform the two attacks against the 13 potentially vulnerable websites and we compare their effectiveness. Attacks are conducted on each website at different times of the day to account for fluctuations in the server's load during daily operations, and the attacks' results are then averaged together. In three websites, both attacks showed an abstention rate of at least 75%, meaning that the computed timing observations cannot be exploited by existing remote timing attacks, i.e., the two classes are too hard to discriminate in practice. Table 3 reports the average performance measures of the two attacks on the remaining ten websites. Our attack outperforms BakingTimer: not only it has a much lower abstention rate (0.06 vs. 0.37), meaning that it can be more broadly applied in practice, but it also has superior true positive rate (0.83 vs. 0.78) and true negative rate (0.93 vs. 0.92). Since we performed the same number of experiments for all websites, there is no single case outweighing the others and biasing the results in our favor. The performance of our attack is generally superior to that of BakingTimer across all our case studies, with the notable exception of LinkedIn, the only website where BakingTimer performs consistently better than our attack. After careful investigation, we observed that this might be explained by the fact that LinkedIn uses a microservice architecture [23]. This indicates that there is a significant probability that distinct servers are responsible for processing authenticated and unauthenticated requests. However, our estimate of the load factor is computed by considering just unauthenticated requests, meaning that our estimate of ρ may be incorrect for authenticated requests. This problem can likely be fixed by extending our attack to estimate two load factors ρ_1, ρ_2 for the different request classes.

6.4 Number of Requests

The number of requests to send is an important requirement for the practicality of a remote timing attack. Prior work in the field considered the number of requests as a hyper-parameter to set a priori, while our attack is designed to stop after a sufficient number

Table 3: Comparison of BakingTimer and our attack on key performance measures.

Measure	BakingTimer	Our Attack
TPR	0.78	0.83
TNR	0.92	0.93
AR	0.37	0.06

Table 4: Comparison of the box test with and without the request cap on key performance measures.

	Measure	Box Test	Box Test with Request Cap
HotCRP	TPR	1.00	1.00
	TNR	1.00	1.00
	AR	0.39	0.26
WordPress	TPR	1.00	1.00
	TNR	1.00	0.84
	AR	0.44	0.34

of requests have been sent to draw reliable conclusions. This complicates a fair comparison with competitors. In our experimental evaluation, the number of requests sent by the box test attack and BakingTimer has been set to 50 and 40, respectively. To assess how our competitors would fare with a number of requests similar to our attack, we estimate their performance using the same number of requests required by our attack to reach a conclusion.

Tables 4 and 5 compare, respectively, the average accuracy of the box test and BakingTimer attacks before and after capping the number of requests. BakingTimer's performance significantly declined and the attack became ineffective, as it not only lost accuracy, but also abstained from drawing conclusions in over 70% of cases. Such behavior aligns with findings reported in the original paper [25], which observed a substantial decrease in performance with a lower number of observations. This outcome is reasonable, as BakingTimer relies on a statistical test that requires several observations to produce reliable results.

Conversely, the box test's abstention rate decreased, though at the cost of some performance loss. This is probably because, when only a subset of measurements is considered, the interval defined by the two quantiles is likely smaller than that computed using the full set of measurements. A smaller interval is prone to overlapping with only one of the two intervals defined by the two baselines, particularly if the baselines' intervals are close to each other. This could explain the decrease in the abstention rate, while the reduction in precision is presumably due to the decreased reliability associated with considering only a limited number of measurements. Consequently, especially when the baseline intervals are close to one another, this can increase the likelihood of errors.

6.5 Robustness Analysis

To assess the robustness of our proposed attack against factors such as network jitter and distributed server architectures, we conduct two additional experiments. These experiments are performed in

Table 5: Comparison of BakingTimer with and without the request cap on key performance measures.

Measure	BakingTimer	BakingTimer with Request Cap
TPR	0.78	0.50
TNR	0.92	0.60
AR	0.37	0.71

a controlled environment to isolate specific factors of interest and minimize external interferences. First, we conduct an experiment using our local HotCRP installation with four different loads, following the same approach as in the controlled setting experiments. In this scenario, the actual RTT is negligible, thus to investigate the impact of the network on our model's accuracy we consider a synthetic RTT defined as the sum of a deterministic component and a variable one. As highlighted by the authors of [7], the RTT distribution often exhibits a spike, which we account for by setting the deterministic part to 40ms – approximately the average measured RTT in Europe, as reported in [21]. To ensure the robustness of our approach under different noise conditions we consider two different probability distributions, Gaussian and Log-Normal, to model the variable part of the RTT. The Gaussian distribution represents a scenario in which RTT observations are mostly clustered around the mean, whereas the Log-Normal distribution introduces a long tail, embedding situations where RTT values are more skewed towards larger values. We set the mean of the Gaussian distribution to 0, and the logarithm of the scale parameter of the Log-Normal distribution to 0.5. The remaining parameters for both distributions are selected to obtain standard deviation values of 0.3 ms, 7 ms, and 21 ms. In particular, we shift Log-Normal observations to obtain a zero mean. Standard deviations in real-world measurements are usually small, as shown in the dataset referred in [21], which is the rationale behind the experiment with standard deviation 0.3 ms. To further assess the robustness of our model, we also evaluate scenarios with significantly larger standard deviations, even though these values exceed those commonly observed in real environments. As shown in Table 6, our method consistently achieves accurate predictions across all tested standard deviation values, experiencing a slight decline in accuracy only for the highest tested value. This is due to the model's ability to effectively distinguish response time patterns across different request types, and the inclusion of an RTT with reasonable variability does not clutter these patterns beyond the model's capabilities.

In the second experiment, we simulated a distributed system with a load balancer by duplicating our HotCRP installation in two different servers and randomly sending requests to one of the two servers when estimating the response times. To isolate the experiment, both servers and the attacker's computer were placed within the same local network. The two servers had different hardware configurations (AMD Ryzen 5 5600G with 32 GB RAM vs. Intel Xeon Gold 5220R with 16 GB RAM), and one server was subjected to a simulated load. The experiment was repeated with four different loads obtained by increasing the number of users through the load tester, following the same approach as in the controlled setting

Table 6: Comparison of the simulated *RTT* with varying distributions and variances on key performance measures.

	Measure	Std Dev 0.3 ms	Std Dev 7 ms	Std Dev 21 ms
Gaussian	TPR	1.00	1.00	0.88
	TNR	1.00	1.00	0.98
	AR	0.00	0.00	0.09
Log Normal	TPR	1.00	1.00	1.00
	TNR	1.00	1.00	0.93
	AR	0.00	0.00	0.00

Table 7: Comparison of our attack with and without the load balancer on key performance measures.

Measure	No load balancing	With load balancing
TPR	1.00	1.00
TNR	1.00	1.00
AR	0.00	0.03

experiments. This setup aimed to replicate the varying response times and workloads that real-world distributed systems experience, effectively modeling a challenging scenario where the load factor is not evenly distributed on the different machines. Table 7 presents the results of our attack on both the single-server and multi-server installations of HotCRP, averaged across all loads. As shown, the abstention rate increases only negligibly (+0.03) when experimenting with two servers, demonstrating the generality of our attack and its effectiveness across different architectures. We attribute the effectiveness of our attack to the model's ability to uniformly abstract the two servers within a single representation providing a reasonable approximation of both of them.

7 Related Work

Remote timing attacks received significant attention from the research community. Felten and Schneider showed that it is possible to abuse the performance improvement offered by web caches to determine whether a target user visited a given web page [8]. More recent work similarly abused web caches to reconstruct the geolocation of a target user [14]. Our work does not directly apply to cache-based timing attacks, because the timing side-channel modeled by our framework is the time required for server-side computation. The pioneers of remote timing attacks enabled by the server-side computation time were Bortz and Boneh, who presented different attacks against web applications [3]. More recent work generalized their ideas to cover additional attack vectors introduced in modern browsers [19, 28] and proposed new timing attacks designed to more effectively breach users' privacy [10, 25]. A key difference of our work with respect to existing attacks is the use of formal modeling to deal with the complexities of remote timing attacks, as discussed in Section 3. As a matter of fact, our experimental evaluation showed that the effectiveness of empirical state-of-the-art attacks like BakingTimer [25] is sub-optimal. Our attack is more robust to variations in the load factor of the target web application and significantly more precise even when tested against real-world websites.

The only paper taking a more rigorous look into the formal theory of remote timing attacks is the seminal work by Crosby et al. [7]. Here, the authors recognize the importance of accounting for the server load factor when measuring response times, and this impact is evaluated as an additive jitter effect on the measurements. However, the scenario proposed in [7] is significantly different from ours. First, it is assumed that the service time is a known constant for the classes: while this may be reasonable for the kind of observations they focus on, in the case of web services, this is known to be generally untrue because the state of the machine has an impact on the processing time due to the complexity of these services (see, e.g., Figure 4 or [9]). Second, the jitter effect is additive with respect to the service time which is more appropriate for a First-In First-Out scheduling rather than a Round-robin as we consider here. Round-robin is more accurate in describing the real behavior of schedulers in nowadays web servers. In Section 6, we compared our approach with the box test proposed in [7] showing that, for the timing attacks considered in this paper, we provide more accuracy in the class prediction.

8 Conclusion

In this paper, we proposed the use of stochastic models to improve the effectiveness of remote timing attacks. Our approach starts from a queuing model of the target web application to attack and uses Bayesian statistics to infer the unknown class of specific network requests of interest. Compared to prior empirical approaches, our attack is more precise, provides quantitative probability bounds, and requires a smaller number of requests. As future work, we would like to experiment with other types of queuing models besides the M/M/1/PS model considered in this paper. In particular, we plan to further investigate the challenges posed by microservice architectures, by refining our approach for tuning model parameters as previously mentioned. Moreover, we will explore possibilities to enhance our analytical model by incorporating more complex queueing network frameworks. In fact, supporting different types of models would be important to generalize to different web applications and settings, e.g., the attacker may perform preliminary tests to determine what is the most appropriate model of the target web application. We also plan to improve our experimental evaluation by extending its scale and by testing the robustness of our method against different disruptive factors, e.g., to assess the impact of geo-location on the effectiveness of our attack.

Acknowledgments

We thank the reviewers for their constructive feedback, which has greatly contributed to the improvement of this paper. This research was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - Next-GenerationEU CUP N.H73C22000890001 and by Agenzia per la cybersicurezza nazionale under the 2024-2025 funding programme for promotion of XL cycle PhD research in cybersecurity – CUP N.H71J24001710005. The views expressed are those of the authors and do not necessarily represent the funding institution.

References

[1] Devdatta Akhawe, Adam Barth, Peifung E. Lam, John C. Mitchell, and Dawn Song. 2010. Towards a Formal Foundation of Web Security. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*. IEEE Computer Society, 290–304. https://doi.org/10.1109/CSF.2010.27

[2] François Baccelli, Bruno Kauffmann, and Darryl Veitch. 2009. Inverse problems in queueing theory and Internet probing. *Queueing Syst. Theory Appl.* 63, 1-4 (2009), 59–107. https://doi.org/10.1007/S11134-009-9150-9

[3] Andrew Bortz and Dan Boneh. 2007. Exposing private information by timing web applications. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy (Eds.). ACM, 621–628. https://doi.org/10.1145/1242572.1242656

[4] Simone Bozzolan, Diletta Olliaro, Stefano Calzavara, Andrea Marin, Gianfranco Balbo, and Matteo Sereno. 2025. Artifacts. https://github.com/Asterius27/SecPerf-Artifacts. Repository containing all of the artifacts related to this paper.

[5] David Brumley and Dan Boneh. 2005. Remote timing attacks are practical. *Comput. Networks* 48, 5 (2005), 701–716. https://doi.org/10.1016/J.COMNET.2005.01.010

[6] Jianhua Cao, Mikael Andersson, Christian Nyberg, and Maria Kihl. 2003. Web server performance modeling using an M/G/1/K*PS queue. In *Proceedings of the 10th International Conference on Telecommunications (ICT)*, Vol. 2. IEEE, 1501 – 1506 vol.2. https://doi.org/10.1109/ICTEL.2003.1191656

[7] Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi. 2009. Opportunities and Limits of Remote Timing Attacks. *ACM Trans. Inf. Syst. Secur.* 12, 3 (2009), 17:1–17:29. https://doi.org/10.1145/1455526.1455530

[8] Edward W. Felten and Michael A. Schneider. 2000. Timing attacks on Web privacy. In *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security, Athens, Greece, November 1-4, 2000*, Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati (Eds.). ACM, 25–32. https://doi.org/10.1145/352600.352606

[9] Kristen Gardner, Mor Harchol-Balter, Alan Scheller-Wolf, Mark Velednitsky, and Samuel Zbarsky. 2017. Redundancy-d: The Power of d Choices for Redundancy. *Oper. Res.* 65, 4 (2017), 1078–1094. https://doi.org/10.1287/OPRE.2016.1582

[10] Nethanel Gelernter and Amir Herzberg. 2015. Cross-Site Search Attacks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM, 1394–1405. https://doi.org/10.1145/2810103.2813688

[11] Mor Harchol-Balter. 2013. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press.

[12] Gabor Horvath. 2024. A CME-based numerical inverse Laplace transformation method. https://it.mathworks.com/matlabcentral/fileexchange/71511-a-cme-based-numerical-inverse-laplace-transformation-method. MathWorks.

[13] Gábor Horváth, Illés Horváth, Salah Al-Deen Almousa, and Miklós Telek. 2020. Numerical inverse Laplace transformation using concentrated matrix exponential distributions. *Perform. Evaluation* 137 (2020), 1–22. https://doi.org/10.1016/J.PEVA.2019.102067

[14] Yaoqi Jia, Xinshu Dong, Zhenkai Liang, and Prateek Saxena. 2015. I Know Where You've Been: Geo-Inference Attacks via the Browser Cache. *IEEE Internet Comput.* 19, 1 (2015), 44–53. https://doi.org/10.1109/MIC.2014.103

[15] Edward G. Coffman Jr., Richard R. Muntz, and Hale F. Trotter. 1970. Waiting Time Distributions for Processor-Sharing Systems. *J. ACM* 17, 1 (1970), 123–130. https://doi.org/10.1145/321556.321568

[16] Soheil Khodayari and Giancarlo Pellegrino. 2022. The State of the SameSite: Studying the Usage, Effectiveness, and Adequacy of SameSite Cookies. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 1590–1607. https://doi.org/10.1109/SP46214.2022.9833637

[17] Leonard Kleinrock. 1975. *Queueing Systems, Volume 1: Theory*. Wiley-Interscience.

[18] Grafana Labs. 2024. Grafana k6. https://k6.io/.

[19] Sangho Lee, Hyungsub Kim, and Jong Kim. 2015. Identifying Cross-origin Resource Status Using Application Cache. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 12 pages. https://doi.org/10.14722/ndss.2015.23027

[20] Henry H. Liu. 2011. *Oracle Database Performance and Scalability: A Quantitative Approach*. Wiley-IEEE Computer Society Press.

[21] Gonzalo Martínez, José Alberto Hernández, Pedro Reviriego, and Paul Reinheimer. 2024. Round Trip Time (RTT) Delay in the Internet: Analysis and Trends. *IEEE Network* 38, 2 (2024), 280–285. https://doi.org/10.1109/MNET004.2300008

[22] Daniel A. Menascé, Virgilio Almeida, and Lawrence W. Dowdy. 2004. *Performance by Design*. Pearson.

[23] Karan Parikh and Steven Ihde. 2015. From a Monolith to Microservices + REST: the Evolution of LinkedIn's Service Architecture. https://www.infoq.com/presentations/linkedin-microservices-urn/.

[24] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 1–15. https://doi.org/10.14722/ndss.2019.23386

[25] Iskander Sánchez-Rola, Davide Balzarotti, and Igor Santos. 2019. BakingTimer: privacy analysis of server-side request processing time. In *Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC 2019, San Juan, PR, USA, December 09-13, 2019*, David M. Balenson (Ed.). ACM, 478–488. https://doi.org/10.1145/3359789.3359803

[26] Mark J. Schervish. 1997. *Theory of Statistics*. Springer.

[27] Connie U. Smith and Lloyd G. Williams. 2001. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley Professional.

[28] Tom van Goethem, Wouter Joosen, and Nick Nikiforakis. 2015. The Clock is Still Ticking: Timing Attacks in the Modern Web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM, 1382–1393. https://doi.org/10.1145/2810103.2813632

[29] Jóakim von Kistowski, Nikolas Herbst, Samuel Kounev, Henning Groenda, Christian Stier, and Sebastian Lehrig. 2017. Modeling and Extracting Load Intensity Profiles. *ACM Trans. Auton. Adapt. Syst.* 11, 4 (2017), 23:1–23:28. https://doi.org/10.1145/3019596

[30] Weikun Wang, Giuliano Casale, Ajay Kattepur, and Manoj Nambiar. 2018. QMLE: A Methodology for Statistical Inference of Service Demands from Queueing Data. *ACM Trans. Model. Perform. Evaluation Comput. Syst.* 3, 4 (2018), 17:1–17:28. https://doi.org/10.1145/3233180

A Attack Implementation Details

Measurement inaccuracies are an inherent aspect of any empirical analysis, arising from various factors such as limitations in measurement tools, environmental noise, and observer variability. While these errors cannot be completely avoided, they can often be mitigated through careful data processing. Here, we discuss methods used to address these issues. First, we deal with outliers, which may result from measurement errors, rare events, or unique conditions. These outliers can distort statistical analyses, so we iteratively filter out values that deviate from the mean by more than twice the standard deviation. This process continues recursively until all remaining values are within the threshold, ensuring only typical data points are retained.

Another challenge arises from the inability to precisely measure round-trip times (RTTs). In fact, subtracting RTT values from the measurements could lead to zero or negative values because this quantity may not precisely represent the network delay, leading to imprecise or even impossible response time estimates. To address this, we subtract the minimum RTT from all observation vectors (X^0, X^1, X^R, X^A). If any negative or zero values remain, we set them to a minimum threshold of 0.1 to avoid unrealistic results.

B Response Time Distribution in M/M/1/PS

Given a non-negative random variable X, we use $f_X(t)$ and $F_X(t)$ to denote its probability density function (pdf) and cumulative density function (CDF), respectively. The pdf can be expressed in terms of its Laplace-Stieltjes transform (LST):

$$f_X^*(s) = \mathcal{L}(f_X(t)) = E[e^{-sX}] = \int_0^\infty e^{-st} dF_X(t)\,,$$

where E denotes the expectation operator. For instance, if X is exponentially distributed with mean μ^{-1}, its CDF and Laplace-Stieltjes transform are given by:

$$F_X(t) = 1 - e^{-\mu t}\,, \quad f_X^*(s) = \frac{\mu}{\mu + s}\,.$$

Additionally, the Laplace-Stieltjes transform of the CDF can be obtained as $f^*(s)/s$. Knowing either the CDF of a distribution or its Laplace transform is sufficient to characterize it completely, as one can theoretically convert between the two. However, in many practical cases, inverting a Laplace transform is only feasible through numerical methods, requiring specialized algorithms developed for this purpose, as hinted in C.

If the LST $f_X^*(s)$ is known, the probability $Pr\{a \le X \le b\}$ can be computed using:

$$Pr\{a \le X < b\} = \mathcal{L}^{-1}\left(\frac{f_X(s)}{s}\right)(b) - \mathcal{L}^{-1}\left(\frac{f_X(s)}{s}\right)(a), \qquad (10)$$

where $\mathcal{L}^{-1}$ denotes the inverse Laplace transform.

In particular, in this work we leverage the well-known analytical expression for the LST of the waiting time distribution in an M/M/1/PS system, of an arriving job conditioned on the deterministic service time it requires. This expression, originally derived in [15, Eq. (30)], is given by the following equation:

$$F^*_{\mathcal{W}}(\tau, s) = \frac{(1-\rho)(1-\rho r^2)\mathrm{e}^{-\lambda(1-r)\tau}}{(1-\rho r)^2 - \rho(1-r)^2\mathrm{e}^{\frac{-\mu\tau(1-\rho r^2)}{r}}}. \qquad (11)$$

with:

$$r = \frac{\lambda + \mu + s - [(\lambda+\mu+s)^2 - 4\mu\lambda]^{1/2}}{2\lambda}.$$

C Model Implementation Details

In order to address measurement uncertainties, we introduce two thresholds, ϵ and δ, as discussed in Section 5.2 and 5.3. In our experiments, we set $\delta = 0.05$, recognizing that due to measurement errors, the exact service time centroid is unlikely to match the observed value exactly. This threshold defines a small range within which observations are considered comparable to those associated with zero waiting time. The second threshold, ϵ, is used in Eq. (6) to accommodate peaks observed in Figure 3 and to account for measurement uncertainty, allowing us to consider a range of values within which the true response time is expected to lie.

To compute the numerical inversion of the transform described in Appendix B, we employed the Concentrated Matrix Exponential (CME) method, as detailed in [13]. This method is known for its excellent numerical stability and smooth approximation compared to other widely used techniques. Our implementation leverages the robust MATLAB package provided in [12]. The computational time for a single iteration of Eq. (8), considering one combination of class 0 and class 1 parameters, ranges between 120 ms and 180 ms.

An important aspect of our implementation is ensuring physically meaningful outputs, particularly in scenarios where certain conditions cannot occur. For instance, since waiting times cannot be negative, any negative value due to measurement problems that may be given as input to the numerical inversion is automatically set to zero, reflecting the fact that negative waiting times have zero probability of occurrence. Similarly, if measurement errors lead to an estimated load factor of zero, we bypass the numerical transform inversion entirely. Instead, we use an indicator function to assign a probability of 0 or 1, where an observation of waiting time is considered feasible only when it is zero. In this case, we assign a probability of 1, while all other possibilities are deemed physically unrealistic and assigned a probability of 0. Additionally, our computations involve subtracting two values obtained from the numerical inversion, as described in Eq. (6). In cases where the two functions yield nearly identical values, small numerical errors may arise due to floating-point precision, resulting in values close to zero but slightly positive or negative. To address this, we apply a correction by setting differences smaller than 10^{-4} to zero, thereby preventing the propagation of insignificant numerical discrepancies in subsequent calculations.

Finally, for Eq. (9), situations may arise where the two classes are numerically indistinguishable for a given centroid combination, making the computation of Eq. (8) numerically unstable. In such cases, the corresponding centroid combination is excluded, and the final results are normalized accordingly to account for its omission.

D Research Ethics

Our experiments in a controlled setting have no ethical implications, because we only targeted local installations of HotCRP and WordPress. Our experiments in the wild are more delicate from an ethical perspective, but we did our best to minimize harm and ensure that the benefits outweigh the risks. Our cross-site attacks only targeted our own sessions, i.e., we mounted our attacks against ourselves, without breaching the privacy of any web user. All identified vulnerabilities have been responsibly disclosed to the correct points of contact.

E Analyzed Websites

Table 8 reports the full list of websites where we successfully registered an account and logged in using the methodology described in Section 6.3, including the potentially vulnerable ones.

Table 8: Analyzed websites

Domain Name	Tranco Rank	Potentially Vulnerable
google.com	1	No
microsoft.com	2	Yes
facebook.com	3	Yes
youtube.com	8	No
twitter.com	14	Yes
instagram.com	15	No
cloudflare.com	16	No
office.com	19	Yes
linkedin.com	20	Yes
live.com	22	Yes
amazon.com	28	No
wikipedia.org	32	Yes
bing.com	36	Yes
github.com	46	No
pinterest.com	50	Yes
adobe.com	56	Yes
goo.gl	58	No
spotify.com	60	Yes
vimeo.com	61	Yes
skype.com	64	Yes