Time and Time Again: Leveraging TCP Timestamps to Improve Remote Timing Attacks

Vik Vanderlinden
DistriNet, KU Leuven
vik.vanderlinden@kuleuven.be

Tom Van Goethem
DistriNet, KU Leuven
tom.vangoethem@kuleuven.be

Mathy Vanhoef
DistriNet, KU Leuven
mathy.vanhoef@kuleuven.be

***Abstract*—One of the most well-known side-channel attacks is to infer secret information from the time it takes to perform a certain operation. Many systems have been shown to be vulnerable to such attacks, ranging from cryptographic algorithms, web applications, and even micro-architectural implementations. Exploiting these side-channel leaks over a networked connection is known to be challenging due to variations in the round-trip time, i.e., network jitter. Timing attacks have become especially challenging as processors become faster, resulting in smaller timing differences, systems become more complex, making it more difficult to collect consistent measurements, and networks become more congested, amplifying the network jitter.**

In this work we introduce novel remote timing attack methods that are completely unaffected by the jitter on the network path, making them several times more efficient than timing attacks based on the round-trip time, and allow for smaller timing differences to be detected. More specifically, the execution time is inferred from the TCP timestamp values that are generated by the server upon acknowledging the request and sending the response. Furthermore, we show how sequential processing of incoming requests can be leveraged to inflate the time of the secret-dependent operation, resulting in a more accurate attack. Finally, through extensive measurements and a real-world case study we demonstrate that the techniques we introduce in this paper have various advantageous properties compared to other timing attack methods: few(er) prerequisites are required any TCP-based protocol is subject to these attacks, and the attacks can be executed in a distributed manner.

I. Introduction

In today's world, the majority of applications run on an Internet-connected server, whether directly exposed or as part of the backend of a larger service. For users this is great as they can perform banking operations, interact with their friends on social media networks, and perform most of their job tasks online. However, this also poses a significant t hreat a s these servers, which hold all kinds of sensitive information, may also be connected to by adversaries. There are a myriad of ways in which attackers can try to steal the information directly. As these direct attacks, such as SQL injection, are well-known and well-studied they are becoming less prevalent as a result of frameworks that eliminate whole classes of vulnerabilities.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230893
www.ndss-symposium.org

A lesser-known class of attacks, in particular among application developers, are side-channel attacks. One of the most prevalent side-channel attacks is a timing attack, where the time it takes to perform a certain operation leaks information about a specific secret. Compared to other types of attacks, there are relatively few reports of timing attacks against popular services, e.g. in publicly disclosed bug reports on vulnerability reward program (VRP) platforms. One plausible reason for that is that timing attacks are difficult to exploit in practice, primarily as a result of network jitter. Poor accuracy when executing a timing attack also makes it more difficult to detect such leaks in a black-box setting as a high number of requests are needed in order to validate the timing leak.

Although the adversary may try to reduce the impact of the network jitter by sending requests from a host that is located within the same datacenter as the targeted service, this makes attacks significantly more complex and may not always be feasible. The majority of widely used services have their own data center, or serve content through a distributed Content Delivery Network (CDN), from which the attacker can obviously not send requests. The main approach to overcome the effects of jitter is to collect a large number of measurements and then apply statistical methods to detect timing differences in the processing time. A major downside of having to collect many measurements is that the time required to conduct a successful attack significantly increases. Furthermore, network parameters or congestion might also change over time, making it even more difficult to conduct the timing attack. Another disadvantage of the requirement to collect many measurements is that services may implement a rate limiting system, which could even completely thwart timing attacks.

As network jitter is the largest contributor to noise of the measurements, researchers have recently looked into performing timing attacks that are independent of network jitter. More specifically, Van Goethem et al. introduced "timeless timing attacks", which leverage coalescing of two requests and parallel execution [1]. By observing which of the two requests generates a response first, the attacker can infer which request took the least processing time. As the two requests arrive simultaneously at the server, both are affected by the exact same jitter on the network path, thereby making the attack performance unaffected by network jitter.

The timing attack techniques that we present in this paper are similarly unaffected by network jitter.

Conceptually, these techniques are fairly straightforward, yet, to the best of our knowledge, novel: the execution time is directly inferred from the TCP timestamps. The first timestamp is generated by the server when it sends the `ACK` packet that acknowledges the receipt of the request. The second timestamp is included in the actual response, which can only be sent once the request has been completely processed. Although the TCP timestamp values are randomized for each connecting host, they are increased at the same rate as the actual time does (in millisecond, or more recently possibly even microsecond granularity).

In contrast to the timeless timing attacks, which require coalescing of requests and parallel processing of the requests, the prerequisites for TCP timestamp-based attacks are less restrictive. The only two requirements for the basic version of the attack is that the TCP timestamps option is enabled on the server and that a response, typically in the form of an `ACK`, is immediately sent after receiving the request. Based on our measurements on Internet-facing web servers, we find that 88.38% of these are (potentially) susceptible to our attacks.

To detect timing differences of a smaller granularity than those of the timestamps (1 ms or 1 µs), we introduce a method that allows the timing difference to be amplified. In essence, multiple requests are sent simultaneously and then processed sequentially. By measuring the time between the arrival of the first request and the response to the last request, the timing leak is amplified by the number of requests that were sent, while still being unaffected by network jitter.

Our results show that our attack detects timing differences that are 5 to 33 times smaller, and require between $1/5$ and $1/50$ of the number of requests compared to round-trip time (RTT) timing attacks. Through our formal model, we present another advantage of our timestamp-based attacks, which is that they can be launched in a distributed manner, where multiple hosts can attack the same service and then combine their timing measurements. This is possible because network jitter does not affect the measurements and reduces the risk of detection, as the requests originate from a large number of sources, allowing an adversary to circumvent rate-limiting systems.

In summary, we make the following contributions:

- We present two new attack methods that use TCP timestamps to infer server runtimes. We discuss the threat model, formal model, and attack methods in Section III.
- We evaluate to what extent real-world servers can be susceptible to these attacks by testing the support for TCP timestamps and other prerequisites in Section IV.
- By performing measurements on our lab-controlled servers, we demonstrate that TCP timestamp-based attacks can detect 5 to 33 times smaller timing differences and requires 5 to 50 times fewer requests compared to RTT-based timing attacks in Section V.
- We demonstrate the practicality of the introduced attacks through case-studies of a cryptographic attack we reproduced (and reported) against an open-source TLS library, a reproduced attack against OpenSSH and a reproduced attack against ProFTPd in Section V-E.

II. Background and Related Work

A. Timing Attacks

Timing attacks have been around since Kocher published their paper in 1996 [2]. He created a clever attack that allowed him to extract keys used in cryptographic operation simply by measuring the time it took for an operation to be completed. That time could be measured accurately by attaching probes to a device that executed the operations, such as a smartcard.

For a long time it was thought that these attacks were only possible on devices with low computational power, where the operations are executed slower, and not over a network, where the sensitive operations would be executed on a powerful and faster server. Additionally it was only possible to take remote noisy measurements over a network as opposed to local accurate ones directly from a device. Often, connections over the Internet are noisy and may change over time, making it very difficult to measure accurate runtimes of a server-side operation on the opposite side of the connection [3]. Despite these difficulties, Brumley et al. showed in 2003 they could execute timing attacks over a network, albeit a local network where latency and noise are usually low in comparison with the Internet [4].

Timing attacks became more prevalent when Bortz et al. defined two attack variants in their influential work: direct and cross-site attacks. Direct attacks are executed from an attacker machine to a victim server directly. For cross-site attacks, the attacker serves a victim user with a malicious piece of JavaScript code that executes the timing attack from the context of the victim‘s browser [5]. In further work, Crosby et al. tested multiple methods of analyzing collected RTTs and show that their then newly created ‘box test’ performs best [6]. Researchers continued to exploit cryptographic libraries or implementations of OpenSSH and TLS, now over remote connections [7], [8], or showed that timing attacks can also be used on or against Wi-Fi networks [9], [10], [1]. Others continued to analyze the statistical methods employed for analyzing timing attack data [11], and some focused on measuring time accurately from within a browser [12], [13] or moved from cryptographic exploits to exposure of private user-information [5], [12], [1], [14], [15], something that was already shown to be effective against a local cache much earlier [16]. More recent work even showed that the location of users can be extracted from timings of SMS messages [17], [18] or instant messengers [19].

Recently, several works introduced new attack mechanisms to reduce the noise in measurements. Van Goethem et al. created a new attack that eliminates transport noise by sending two requests simultaneously, making sure they are processed concurrently, and observing which response was sent from the server first, and thus is likely to have the shortest execution time [1]. Vanderlinden et al. used the ‘server-timing’ header to measure runtime directly [20], and later created an attack in which the client attempts to synchronize to the server’s

'date' response header values, to then use the value of this header to infer whether requests took longer to be processed [21]. This reduces noise but still creates challenges as up- and downstream latencies may differ and can be challenging to measure [22], [23], [24]. Recently, Kettle updates the Timeless Timing Attacks of Van Goethem et al. and explores the practical impact of the updated attack mechanism [25].

Over the years, defenses against timing attacks have been proposed, but their real-world adoption is unclear [26], [27].

B. TCP Timestamps

TCP Timestamps are defined in RFC7323 [28]. Their main components are the `TSval` (**T**ime**S**tamp **VAL**ue) and `TSecr` (**T**ime**S**tamp **EC**ho **R**eply) fields. A timestamp value is set in `TSval` when sending a segment and is echoed back in the `TSecr` in the next segment by the other host. When a host receives any TCP segment after the first `SYN`, they can now subtract the received `TSecr` value (which is the `TSval` value sent in a previous segment) from their current timestamp and calculate the RTT as accurately as their internal clock. This implementation lets the receiver blindly echo the timestamp bytes without interpreting them or knowing their meaning. Using TCP Timestamps has several advantages [28]:

1) For TCP's congestion control algorithms to function correctly, they need to be able to estimate the RTT [29]. Multiple timers in TCP are based on the RTT to be able to determine for instance when to send a retransmission. When the RTT is incorrectly estimated, TCP may send more spurious retransmissions or will wait too long to send retransmissions, resulting in a suboptimal connection.

2) Additionally, Protection Against Wrap-over Sequences (PAWS) was introduced (originally defined in RFC1185 [30]). Because more and more connections are getting significantly faster (specifically high speed networks in the Internet backbone), the sequence numbers used by TCP can wrap around over the largest value back to zero. This could lead to potential issues when a retransmission occurs after the sequence number has wrapped completely around, or when a new connection is started after an interruption and the initial sequence number happens to be in the same range as the old one. By using TCP timestamps, the PAWS algorithm is able to detect whether or not a segment is a retransmission from a previous sequence range or a segment with sequence number wrapped around, thereby preventing any issues.

Besides those two major use-cases, additionally the Eifel detection and response algorithms can be implemented when TCP Timestamps are used, the timestamps can be used for the implementation of the LEDBAT or TCP-LP congestion control algorithms and some other optimizations were created such as the "best current practice" of reducing the `TIME-WAIT` state when TCP Timestamps are enabled [31], [32], [33], [34], [35].

For the RTT measurement (RTTM), LEDBAT and TCP-LP, the rate at which a timestamp value changes should at least approximate the rate of change of the actual time. PAWS, the Eifel detection and response algorithms and the reduction of the `TIME-WAIT` state on the other hand, do not require a timestamp to work, only a monotonic non-decreasing value that increases at least once for every range of sequence numbers [31], [32], [33], [34], [35], [30], [28]. Because some use-cases require a value proportionate to the actual time, choosing a time-based value as the timestamp value is a logical default choice. The exact values of the timestamp values do not necessarily matter, however, previous work has shown that the values of TCP timestamps can be used maliciously.

Separate from the official use-cases defined in RFCs, there are additional works on passive estimation of RTTs using TCP timestamp values [36], [37], [38].

1) Malicious Use: In the beginning of 2001, McDanel sent a message to a mailing list regarding the use of TCP Timestamps for collecting *uptime* information [39]. He explains that the timestamp value is usually reset to 0 upon boot for most systems at that time. Depending on the update frequency of the used clock, he is able to find systems that theoretically may expose the uptime of a system until 24 days (1 ms timestamp) or 34 years (500 ms timestamp) after boot. Although the impact of leaking this information may be limited, a later article from Hailperin in 2015 mentions detecting a successful DoS attack or determining the patch-level as possible use-cases that may be interesting to an attacker [40]. Although RFC7323 from 2014 mentions the randomization of the timestamp value, Hailperin in their article confirm the timestamp could still often be used to detect system uptimes [28], [40].

Besides calculating update, McDanel mentions the use of timestamps for identifying the number of systems in a load balanced environment. By repeatedly making connections to the environment the timestamps from the remote system will change when connected to another machine behind the load balancer [39]. A similar observation was made by Bursztein in Phrack magazine in 2005. They show that it is easy to count the number of hosts behind a system using network address translation (NAT) (as long as the timestamps are not being rewritten by the NAT) and propose a method to identify whether Port Address Translation in the NAT is being used, and whether or not there are multiple hosts behind a port.

In addition to these attacks, they propose options for defenses, the most interesting one is adding a random increment to the timestamps [41]. It is interesting to note that this was a full 10 years before Hailperin confirmed that systems still used 0 as the initial value of timestamps [40]. Wicherski et al. improve upon the idea of Bursztein to identify hosts behind a system using NAT in their 2013 paper and use it to implement software for network filtering based on the TCP timestamps. To their admission, the already proposed update to add a random offset based on each unique connection would circumvent their system [42].

Giffin et al. describe how they create a covert channel using TCP timestamps by using the lowest significant bit of the timestamp to communicate to a receiving party by introducing very small delays in the packet transmission times. While doing this, they assure the timestamps still adhere to the required monotone non-decreasing nature described in RFC1323 (which was the most recent version of the RFC at

the time of their writing) [43], [44].

For any of the aforementioned attacks it is unclear whether they are still viable, although it is likely that the host based randomization of the timestamp clock thwarted most of these attacks.

2) Proposed Updates and Extensions: In the past years, many updates were proposed to the timestamps option. They are worth noting because of their common idea: increasing the accuracy of the timestamps from milliseconds to microseconds.

In 2017, researchers showed that using a millisecond timestamp accuracy is not enough in a datacenter [45]. Due to the high volume of traffic and low RTTs within a datacenter, they claim it would be beneficial to use microsecond accurate timestamps. In the IETF99 meeting, they follow up with an official draft of a new option that can be used to replace the TCP timestamps option when supported by both hosts [46], [47]. The authors show intent to implement the option into the Linux kernel, but the draft expired and was never updated anymore. A few years later, in IETF109, another option was proposed, under the name 'extended timestamps option' (ETSOpt) [48], [49]. Once again, a draft was published that expired and wasn't updated. For both updates it remains unclear whether there are any mainstream implementations of these unofficial options.

More recently in late 2023, the microsecond timestamps were implemented in the linux kernel and are available in version 6.7 and up [50]. The option to use microsecond accuracy was implemented using a route attribute, which can be set using iproute2 version 6.8.0 and up [51]. As an example, these versions of the kernel and iproute are available by default in Ubuntu 24.10, which means developers can choose to enable the microsecond accurate timestamps starting at that version.

C. Comparison with related attacks

While there are multiple other attacks that also aim to reduce the amount of noise in measurements in various ways, we highlight the main differences with our new attack.

Vanderlinden et al. use the 'server-timing' HTTP header to read timing information from the server [20]. They show adoption of the header is low, in contrast with the TCP timestamps option, which is widely enabled as shown in Section IV. In later work, they used the 'date' HTTP header to infer runtime by synchronizing with the server clock and observing the distribution of clock rollovers in responses [21]. By using this technique, they eliminate downstream jitter, but not upstream, which means they still have a dependence on the network path. In addition, this attack cannot be distributed because it (1) still includes jitter from the path to the server in its measurements and (2) first performs a clock sync that is unique to one connection. Both previous attacks are limited to HTTP because of their use of HTTP headers, while our attack is broader since all protocols running over TCP can be exploited (e.g. SSH or FTP, see Section V-E).

As previously mentioned, Van Goethem et al. use the Timeless Timing Attacks to remove all network jitter from timing samples [1]. We highlight differences with the Timeless Timing Attacks throughout this paper.

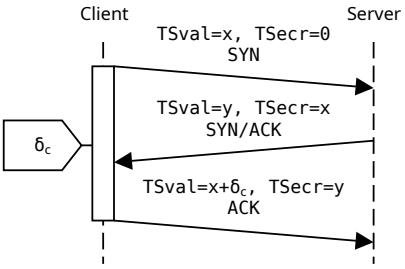

Fig. 1. The TCP 3-way handshake with timestamp option support, where the `TSval` of the client is increased by δ_c between `SYN` and `ACK`.

III. Attack Overview

A. Attack Mechanisms

The key aspect of our proposed timing attack is to utilize TCP timestamps that are set by the server to infer the runtime of processing a certain request. In Figure 1 we show the TCP 3-way handshake for two hosts that have timestamps enabled. The client (the host that opens the connection) sends a `SYN` with (among possible other options) the timestamps option that includes a client-chosen initial value for `TSval` and 0 for `TSecr` because there is no initial value to echo back to the server. If the server (the host that accepts the connection) also supports TCP timestamps, it replies with the option set in the `SYN/ACK` segment with the `TSecr` set to the value that the client sent in the previous segment. The server selects a `TSval` value that is usually based on the current time in milli- or (depending on its configuration) microseconds on the server. After this interaction the timestamps option is configured and the client and server continue to send the `TSval` and reply the `TSecr` to each other. If the server does not support the timestamps option, it will not reply with that option in the `SYN/ACK` segment after which the client host knows to disable the option on their side as well in order to save bandwidth for the continuation of the connection.

The initial value of the timestamp, denoted by x and y in Figure 1 are calculated based on an internal clock to which a random value unique to each connecting host is added [52]. This prevents certain information leakages discussed in Section II. Despite the prevention of information leakage, the randomized initial values do not prevent timing attacks.

1) TCPTS Timing Attack: The basic idea of our attack is to utilize the timestamps that the server sends along with its replies to infer the runtime of a process on the server, as visualized in Figure 2. Because with every TCP segment, including segments containing data like HTTP/TLS responses as well as simple ACK segments, the server includes a new TCP timestamp based on its current time, we can calculate the difference between two subsequent timestamps from the server. Due to the per-host random offset that is added to the timestamps, we cannot discover the local time of the server.

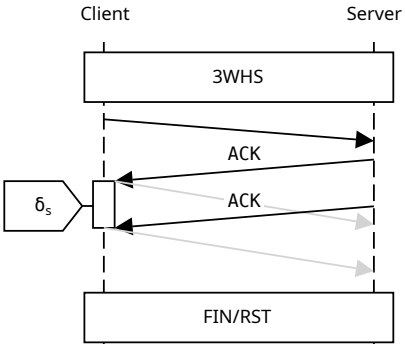

Fig. 2. Basic attack mechanism where the difference between two TCP timestamps δ_s can be used to estimate server runtime.

However, because the timestamps are proportionate to the time, we can infer the runtime of a process on the server.

An important note to make for this attack mechanism is that when sending a request, the server has to send an immediate `ACK` before processing the request and another segment (for instance with HTTP response and `ACK`) after the processing. For each of those two segments, there will be a timestamp option attached with two separate `TSval` values. By subtracting the two timestamps, we obtain an approximation of the runtime on the server, rounded down to the accuracy of the server clock. We explore the real-world immediate acknowledgement behavior in Section IV.

2) Runtime Multiplication Enhancement: A major enhancement to this basic attack can be made when multiple requests are coalesced so they arrive on the server at the same time. This can for instance be achieved by sending one TCP segment with multiple HTTP requests, a scenario used as an example in this section. By coalescing multiple requests into a single TCP segment, all requests will arrive on the server at exactly the same time and will be processed one after another when using `HTTP/1.1`.

In Figure 3 we show the enhanced attack mechanism where the first two `ACK`s show the exact same behavior as in Figure 2. Now, because there are more requests to be processed, the client will receive more responses and thus more timestamps, which can all be used at once to estimate the runtime of the server process.

When sending N requests in one segment, we can calculate the time difference between the initial `ACK`, which is an immediate acknowledgement to the first request, and the timestamp from the last response. As a result, the timing difference is inflated N times, which can be particularly helpful when it is smaller than the granularity of the server's timestamp.

The requirement of an immediate `ACK` as discussed for the basic attack mechanism is not as important for this enhanced attack mechanism. Without an immediate acknowledgment there is still the opportunity to multiply the runtime by $N-1$ instead of N by using the `TSval` value from the response to the first request instead of the immediate `ACK`.

Because the Internet is becoming more and more complex

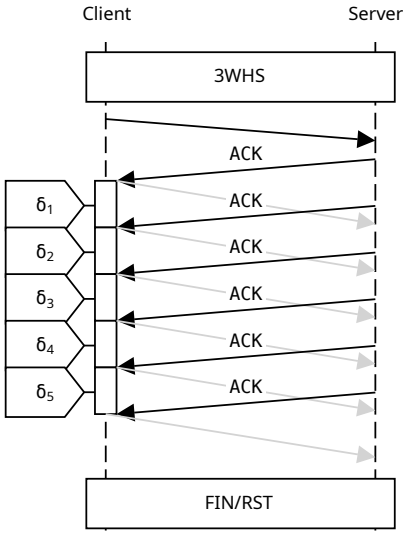

Fig. 3. Advanced attack mechanism where difference between multiple timestamps can be combined to estimate runtime more accurately.

and websites often serve a larger amount of requests to clients, nginx increased the default maximum number of requests that can be handled over a single connection to 1 000 as of version `1.19.10`, before which it was limited to 100 [53]. For Apache 2.4, the default is still set to 100, but can be easily modified through the `MaxKeepAliveRequests` directive [54].

3) Distributed Attack: Because our attack has no dependency at all on the network connection, it is feasible to execute it in a distributed manner. An attacker could let many machines, or even a botnet, each collect a small number of samples and combine these samples into a single dataset from which they can infer the runtime of one endpoint of a server.

This is a powerful property because it allows an attacker to remain undetected even when collecting large amounts of samples. Additionally, the attacker will be able to collect data much more quickly than if they would have to use a single machine, on which the simultaneous collection streams will be limited due to the amount of network traffic an attacker will generate. Sending this much traffic from one machine can have the unintended side-effect of creating congestion on their own connection, which will increase the noisiness of the RTT measurements. Finally, the distributed attack would also allow adversaries to circumvent rate limiting systems imposed by the server, which are typically based on IP addresses. We note that the Timeless Timing Attacks described by Van Goethem et al. can also be executed in a distributed manner, but this is not explored in their paper [1].

B. Formal Model

To clarify the attack and its advantages, we introduce a formal model of the attack mechanisms. We follow the model of Crosby et al. [6] and split the RTT, denoted by R, into the server processing time P and the transport time over

the network T. In a classical timing attack, where response times over the network are directly collected and analysed, the measurements obtained are of the form $R = P+T$. Here, both the server processing time P and the network transport time T consist of sub-operations, such as the transport times between each middlebox on the Internet, as well as the server reading the data from the request, parsing and processing it, and so on. An important sub-operation is the sensitive operation we want to time in our attack, whose duration we will represent by P_M. The other components can be split into the sub-operations $m \in [1, \ldots, M-1]$ before the sensitive operation and the sub-operations $n \in [M+1, \ldots, N]$ after the sensitive operation. This leads to the following equation:

$$R = \sum_{m=1}^{M-1} (T_m + P_m) + P_M + \sum_{n=M+1}^{N} (P_n + T_n) \quad (1)$$

In our novel attack, we will measure the time that has passed between the generation of two TCP timestamps. More precisely, we will measure the time between: (1) the timestamp generated by the server right before processing a request; and (2) the timestamp generated right after processing the request, i.e., when the response is being sent. This means we only measure the sub-operations $m \in [k, \ldots, M-1]$, the sensitive operation P_M, and sub-operations $n \in [M+1, \ldots, \ell]$, where $1 \leq k < M$ and $1 < \ell \leq N$. All of the operations within our measurement are happening on the server, so we can say that T_m and T_n are equal to zero and can be removed from the equation. In addition, we define i to be the identifier of the request that is processed over the connection, leading to:

$$R_i = \sum_{m=k}^{M-1} P_{m,i} + P_M + \sum_{n=M+1}^{\ell} P_{n,i} \quad (2)$$

This equation describes exactly the amount of time between the generation of two timestamps. The time that can be measured in our basic attack is this value for the first request of the connection, floored to the accuracy of the TCP Timestamps, namely 1 ms (or optionally 1 µs). With S denoting a sample that our attack measures, this gives $S = \lfloor R_0 \rfloor$.

For the enhanced attack mechanism, we force C requests to arrive at the server simultaneously and measure the time between the start of processing the first request and the sending of the response to the last request. In essence, this is equivalent to summing Equation (2) C times, once again this value will be floored to the accuracy of the timestamp, giving $S = \left\lfloor \sum_{i=1}^{C} R_i \right\rfloor$ which can be further worked out as:

$$\begin{aligned} S &= \left\lfloor \sum_{i=1}^{C} \sum_{m=k}^{M-1} P_{m,i} + \sum_{i=1}^{C} P_M + \sum_{i=1}^{C} \sum_{n=M+1}^{\ell} P_{n,i} \right\rfloor \\ &= \left\lfloor \sum_{i=1}^{C} \sum_{m=k}^{M-1} P_{m,i} + C \times P_M + \sum_{i=1}^{C} \sum_{n=M+1}^{\ell} P_{n,i} \right\rfloor \end{aligned} \quad (3)$$

In the expansion of Equation (3) we can see that the processing time is multiplied by C, which means that a small (secret-dependent) processing time may be multiplied to a duration that is large enough to be detected in the floored value captured by the attacker. Furthermore, when looking at the individual components in the final equation, these consist of C times the processing time before and after the secret operation and C times the processing time of the secret operation. These components consist of a true processing time and a variation, i.e., jitter, on the processing time. This means that although the timing difference from the secret operation is amplified (by C times), so is the jitter of the operations that take place on the server. However, because those operations are local on the server, the jitter associated with these operations is several orders of magnitude smaller than the jitter incurred on the network path. As a result, our novel timing attack is as performant as if it were conducted from the local machine since it is unaffected by jitter on the network path.

C. Threat Model

We assume the attacker is remote and launches attacks from any Internet-connected device, or several devices in the case of a distributed attack. This is in contrast with most (traditional) timing attacks where the attacker ideally positions themselves as close as possible to the victim host. Here, "close" refers to the latency and number of hops between the attacker and the victim hosts. Geographically distant servers may thus be considered closer or more optimally positioned than other, geographically nearby hosts depending on the network connection (e.g. inter-datacenter vs. from a residential network). Not having the requirement to be close to the targeted server provides the advantage that attacks can still be effective even though it is not possible to be in the same data center as the victim, or when the server is hosted on-premises.

In our basic attack scenario an attacker has to be able to read all TCP response segments in order to collect the TCP timestamps. In addition, when executing the more enhanced variant of the attack, the attacker has to be able to coalesce multiple requests into one TCP segment. Given that the initiated requests are under the control of the attacker, this prerequisite can be trivially achieved. Furthermore, the requests sent by the attacker should be handled sequentially by the server; for web requests this means that HTTP/1.1 should be used in favor of HTTP/2 (most servers that support the latter also support the former), and in most other protocols the default is sequential execution. Adding to that constraint, the server should not close the connection immediately after each request for the enhanced attack to properly function, as it relies on the measurement of timestamp values in many response segments. We found that there are a few cases where certain web servers would send a `Connection: close` response header after each request. In Section IV-C we explore in more detail to which extent these prerequisites are met in real-world applications.

In addition to direct timing attacks, i.e. those where the attacker directly sends requests to the victim server, one could also consider leveraging these attacks in a cross-site context [5]. This means that the requests are initiated by the

Attack	Precondition	Servers tested	Servers supported	Servers %
Both	TCP Timestamps Option	883 892	785 778	88.90%
Single	Immediate Acknowledgment	613 909	610 224	99.40%
Multiplied	Persistent Connections	613 909	586 609	95.55%
Multiplied	Runtime $>$ 1 ms	557 215	386 381	69,34%

TABLE I
AN OVERVIEW OF ATTACK PRECONDITIONS AND THEIR OCCURRENCE ON THE INTERNET. THE BOTTOM THREE MEASUREMENTS WERE PERFORMED ON A SUBSET OF DOMAINS FROM THE SET OF DOMAINS THAT SUPPORT TIMESTAMPS AND THEREFORE SHOW THE SUPPORT OF EACH FEATURE FOR TIMESTAMP-ENABLED DOMAINS

user's browser while they are visiting a malicious site. As a result, the requests would carry the user's authentication information for that website, allowing the attacker to infer information specifically for the user. Because the attacker cannot observe TCP traffic from within the browser, they need to position themselves in an active Machine-in-the-Middle (MitM) position, allowing them to observe network traffic from and to the victim host. Note that the traffic between the victim and targeted server can still be encrypted, as the attacker only needs to read out the (unencrypted) TCP options. From the MitM position, which can be achieved in various ways such as by setting up a rogue Wi-Fi access point, being a privileged network administrator, or executing ARP or DHCP spoofing, the attacker can also inject malicious JavaScript to trigger the requests. This can be done against any connection that is (initially) set up over an insecure channel. Although most HTTP/1.1 servers support HTTP request pipelining, browsers will wait for a response before sending another requests on the same connection. As such, only the basic attack can be performed in the cross-site attack scenario.

In a cross-site attack, the attacker has limited control over the network conditions and location of the victim. In particular when using a MitM position while relaying the network traffic to their destinations, there will likely be significantly more network jitter on the connections than in the evaluation setup we used to test the limits of our attack. This means that, because our attack is independent of network jitter, it can be expected to be equally performant as in our evaluation setup, but the same is not true for the classical attack. With increasing network jitter, the classical attack can be expected to deteriorate while our novel attack can still operate at the same performance.

IV. SUSCEPTIBILITY

In order to demonstrate the potential for widespread applicability of our attacks, we present an overview of some preconditions and test the susceptibility of these preconditions on the Internet. For each of the preconditions, we used a subset of the Tranco list [55]. [1] An overview of the preconditions and their use can be found in Table I.

[1]Available at https://tranco-list.eu/list/KJ33W

A. Timestamp Option Support

A requirement to perform our attacks is that the victim has TCP timestamps enabled, since an adversary can always enable TCP timestamps on their device. Fortunately, we found that the timestamp option is enabled by default on both Linux and Windows Server, though it is disabled by default on Windows Workstation 10. This means that for users using Windows as their operating system, a cross-site attack will not be possible unless they enable the timestamps option manually or the attacker finds a way to do so automatically. When the option is enabled, the attacker's kernel is instructed to send out the timestamps option in each outgoing TCP handshake (initial `SYN`) and attach the option to each TCP segment. The server still has the option to either include a timestamps option in their response or not. This means the attacker will always be able to enable TCP timestamps on their own device, but they still rely on the server for the attacks to work. In a cross-site attack scenario, the attacker does not have complete control over the client device. They have to rely on the configuration of the victim's device in this case.

To estimate the number of servers on the Internet that enable TCP timestamps, we crawled the top million domains in the Tranco list and tried to send each a TCP 3-way handshake on ports 80 and 443, including the timestamps option. After the handshake completes, we followed up by sending a TCP `RST` segment to release the connection. From this interaction, we can determine whether the server supports TCP Timestamps by checking whether or not they responded to our `SYN` with a `SYN-ACK` that includes the timestamps option.

We disregard domains that cause an error and find that almost 90% of servers support TCP timestamps when randomly reaching out to them. The number of domains supporting timestamps on port 443 is practically the same. This makes sense because we will often end up on the same machine for both ports and the kernel will attach a timestamp regardless of the transport-level port. We can conclude that attackers have a reasonable chance that their target supports TCP timestamps.

To check whether there are any servers that use a timestamp with a granularity lower than a millisecond, we performed another scan in which we opened a connection to the server by sending a `SYN` segment with a timestamp option attached, waiting for one second and then sending a `FIN` segment. By comparing the difference in timestamp value (if supported and enabled by the server) in the `FIN/ACK` and `SYN/ACK` against the sleep timeout of one second, we can roughly estimate the timestamp granularity. Out of 730 557 tested domains, we find that only approximately `50` appear to use timestamps that have a higher accuracy than one millisecond. Further manual inspection reveals that some hosts indeed use timestamps that are more accurate than one millisecond, but the majority of results are false positives, for instance due to the fact that these hosts replied with a `TSVal` value of 0 in their `SYN/ACK` segment, which results in an artificially large difference between the two timestamp values. These systems do not seem to start counting at 0, which was the default

way of using timestamps a long time ago, but they seem to only start sending their actual timestamp values after the three-way handshake is completed. This has no practical impact on our attacks, as we do not use the timestamp included in the `SYN/ACK` in the attack.

B. Immediate Acknowledgments

For the basic attack method to work, the server has to respond to incoming requests twice: once immediately when the request arrives (an 'immediate acknowledgment') and once after the application has processed the request. An attacker has to be able to trigger this behavior from the server to be able to measure the runtime based on the timestamps.

For Linux-based servers (by far the most common OS for servers [56]), a one-time immediate `ACK` is sent when the connection has been established [57]. For subsequent data that is sent, an acknowledgment is sent immediately when more than a full frame has been received (as per the `__tcp_ack_snd_check` function). An attacker could force this condition to always be true by including bogus data in their request (e.g. as part of an unused HTTP request header).

To evaluate how prevalent the immediate `ACK` is in practice, we selected a random sample based on resource and timing constraints from those that have TCP timestamps enabled. For each domain we visited the first page after following all initial redirects (to HTTPS and possibly a landing page). For the $613\,909$ hosts that responded, we find that over 99.40% send an immediate `ACK` after the first request on a new connection, which matches the behavior defined in the Linux kernel (which is most likely similar to kernels of other OSes). From this high percentage, we can conclude that the basic attack will work against almost all servers that have TCP timestamps enabled.

C. Persistent Connections

Our stronger attack method that leverages runtime multiplication has different preconditions compared to the basic method. In order for the attack to work, a server has to accept more than one request over a single connection. When using HTTP/1.1, the default behavior is to keep connections alive so subsequent requests can be sent directly over an existing connection. This is more efficient than closing each connection because setting up a connection takes time and resources. Although by default connections remain open, both the server and client have the option to indicate they want the connection to be closed. By adding the `Connection: close` header to a request or response, they can communicate that the connection should be closed after the next response is received.

A server that sends a `Connection: close` header will in general not respond to subsequent requests over the same connection anymore. It may mean that the server does not support persistent connections, so it is best to honor this header to avoid unexpected side-effects. In our test with $613\,909$ servers, we find that less than 4.5% of servers respond with a `Connection: close` header after the first request. That means more than 95.5% of servers support persistent connections and are potentially vulnerable to the runtime multiplication timing attack.

D. Server Runtimes

1) Problem Statement: Our attack must handle TCP's congestion and flow control methods, since they may otherwise interfere with the attack. A first mechanism is the initial congestion window (icwnd), which prevents a host from sending a large batch of data right after creating a connection. The congestion window slowly opens up and allows the host to send more and more data whenever an acknowledgment from the receiving side makes its way back. This mechanism would be detrimental to our attack because an attacker would not able to send a large number of requests at once to the victim. However, because the initial congestion window is a parameter of the sender, the attacker can freely alter this parameter, removing this obstacle. For the basic attack method the icwnd is not important because only one request is sent.

A second mechanism is the receive window that is controlled by the receiving side. Upon sending data, the receive window of the receiver is filled up gradually. When the window is full, the server instructs the client to stop sending more data by advertising a zero window size. The client, i.e., attacker, now has to wait, otherwise data will be lost as the server can't receive it anymore. Once enough data has been processed to open up the window, the server will increase the receive window size and advertise this in one of the responses that are being sent back (or in another window size advertisement). Now the client is able to send more data, but overall, the time lost is at least one RTT (at the moment the server sends the updated receive window, that response has to travel to the attacker, which then has to send their next batch of requests). If the remaining requests in the buffer together have a processing time smaller than the RTT, the server will have to stop processing requests and wait for the next batch to arrive, which is a problem for the attack. If this occurs, the total processing time of all requests will depend in part on the RTT, which is exactly what we attempt to avoid.

2) Solution: The effect only occurs when an attacker wants to send more requests to the server than fit into the receive window. One way to make sure this doesn't happen is to check the server's window size when opening the connection, and then never sending more data than the server advertises it can receive. When an attacker wants to send more data than fits into the window, they have two options: they can either send the next batch of requests at the moment they expect the server's receive window to open up, or they can wait for the server to send the updated window size. The latter can be used by an attacker when the processing time of the requests is large enough such that the next batch of requests can reach the server before that server has run out of requests to process and has to wait for the next batch of requests.

Once again, we reached out to the homepages of a random sample of $557\,215$ domains from the Tranco list that support TCP timestamps to estimate if this would be an issue on real-life websites. By using the TCP timestamps to measure

the runtime, we get an accurate representation of the time it took the server to process the request. We find that out of over 386 000 domains, just under 70% take more than 1 ms to process the request. This is in line with what can be expected, as many of these homepages are likely using caching mechanisms to speed up the processing of the requests. When executing timing attacks against other endpoints on a site (such as endpoints where the site has an interaction with a database or other backend server such as an authentication service), we can safely assume that the overall runtime will be significantly larger. Therefore, we conclude that the majority of endpoints an attacker would be interested in will not be affected by the receive window size problem.

E. Coalescing Requests

In our attack that leverages request coalescing, a prerequisite is that the requests arrive simultaneously at the server. If there is any delay between requests being processed, network jitter would be introduced, decreasing the attack performance. There are several options for coalescing requests, examples being:

- **Single TCP segment.** The attacker can send multiple requests in one TCP segment, but the multiplication factor is limited by the maximum segment size (MSS) of the connection, which is typically around 1 460 bytes.
- **TLS record.** Multiple requests can be combined into a single TLS record. The server can only decrypt and process this TLS record once it has been received entirely. The maximum size of a single TLS record is 16 384 bytes.
- **Omitted TCP segment.** An adversary sends requests over multiple TCP segments, and sends the first TCP segment last over the wire. Only when the missing first segment arrives can the server process all requests. The limit depends on the maximum receive buffer defined by the server. For example, on AWS Ubuntu-based servers, this maximum value is set to 6 MiB by default, which is more than sufficient to send the requests.

A second requirement is that requests are processed sequentially. Although HTTP/1.1 does not guarantee this [58], we confirmed that nginx, Apache, and Microsoft ISS handle HTTP/1.1 requests sequentially. Overall, for HTTP/1.1, the main limiting factor is how many requests can be sent in a single connection (nowadays up to 1000, recall Section III-A2).

F. Infrastructure Considerations

When compared to prior work on timing attacks that are unaffected by network jitter, namely the Timeless Timing Attacks [1], the prerequisites of our attack are different. The timeless timing attacks require an HTTP/2 connection over which requests are sent that arrive simultaneously at the server and are then processed in parallel. HTTP/1.1 does not support multiplexing of requests over a single connection, so, to the best of our knowledge, there is no other remote timing attack that is unaffected by network jitter that has a comparable performance to our proposed technique using TCP timestamps.

For the timestamp-based attack that requires coalescing of requests to amplify the timing difference, a requirement is that these requests are processed sequentially. As such, coalescence-based attacks over HTTP/2 are not feasible, as the server will process these requests in parallel, but connections supporting HTTP/1.1 are affected. The Web Almanac of 2024 shows that around a fifth of homepages only support HTTP/1.1, but this is not an issue for our attack: even when HTTP/2 is supported by a server, it is likely that HTTP/1.1 is still supported for backward compatibility [59].

In addition, when visiting many sites on the Internet, the origin server (actual host of the site's content) will not be reached but rather a CDN server can be found. Many sites either use a CDN for caching static content or as a proxy that adds some protection against e.g., Distributed Denial-of-Service (DDoS) attacks. Looking at the Web Almanac data, it is clear that many publicly facing sites support HTTP/2 [59]. Large CDNs are likely to implement new standards faster than individual websites. Once they do, the adoption of for instance HTTP/2 or even HTTP/3 rises fast because of the large number of sites using these CDNs as the frontline of their infrastructure. We tested the top three CDNs (Cloudflare, Amazon CloudFront and Google Cloud CDN) and found that, as expected, they do not pass through TCP timestamps from the origin server. This means the attack cannot be carried out through the CDN and must instead target the origin server directly. When sending traffic over the Internet, we never observed that timestamps were rewritten or dropped by middleboxes such as NATs, routers, firewalls, Intrusion Prevention Systems (IPSs), or similar devices.

The high adoption of HTTP/2 in the Web Almanac data thus includes many CDN endpoints which are more likely to support HTTP/2 [59]. Because popular web servers nginx and Apache do not yet enable HTTP/2 out of the box, it is likely that relatively fewer origin servers actually support HTTP/2. This means that the Timeless Timing Attacks are less likely to be usable against those origin servers, while our attack would be usable. In addition, we expect that a Timeless Timing Attack against a CDN server will be less performant due to the additional jitter imposed by the backend requests between the CDN server and the origin server.

As mentioned, we cannot perform our attack through a CDN, but must instead target the origin server. Fortunately, Vissers et al. have shown that many origin servers can be found despite them using a CDN [60]. They claim most sites do not use a CDN to hide their origin server, but mainly for performance reasons. An adversary can therefore use the techniques described by Vissers et al. to find origin servers for the victim site and perform the attack against these servers directly, meaning the usage of CDNs does not limit our attack.

V. Results

To evaluate our novel attack methods, we perform measurements on our lab-controlled servers hosted on popular cloud platforms. We collect millions of samples and systematically analyze them to explore the limits of the attack methods. We compare our methods against the well known RTT based timing attack method by collecting measurements against the

same servers to allow for a fair comparison. In addition, we test the attack against three real-world applications. Our datasets have been made public on Zenodo.[2]

A. Experimental Setup

We set up multiple servers on Google GCP in a region on our university's continent to act as the victim servers, and on our internal university cloud located on our campus to act as the adversary machines. Note that the performance of our attack is unaffected by the location of the server; we opted for a nearby server to reduce latency and thereby speed up the experiments, allowing us to perform a comparability analysis with a high number of measurements. All servers have high speed Internet connectivity of at least 1 Gbps. Our victim servers were large instances of type C4 with a constant clock frequency (through the `all-core turbo frequency` option) to prevent a varying clock frequency from interfering with the runtime of the server-side scripts. The virtual CPUs remain shared with other tenants, which could still be a source of noise in the execution. Our web servers are set up to respond to our requests with a variable delay that is configurable through a request query parameter.

On the adversary servers, we increased the initial congestion window such that we would not be limited by the congestion control algorithms of TCP. We disabled Nagle's algorithm (by setting the `TCP_NODELAY` flag for each connection) to make sure we could send requests without any delay added by the kernel. Finally, we also disabled the TCP segmentation offloading features of each network interface card (NIC) to be able to correctly identify each of the received segments exactly as they were sent by the servers, without the NICs merging the segments before they could be captured. These are all settings that can be changed by any attacker executing a direct timing attack as long as they have root privileges on the machine they are launching the attacks from.

We collected $35\,000$ samples consisting of a total of approximately 100 million requests to the HTTP servers for each timing difference that we aimed to detect: 500, 250, 100, 75, 50, 25, 10, 5 and $1\,\mu s$ for the millisecond timestamps and $5\,000$, $1\,000$, 750 and 500 ns for the microsecond timestamps. This results in an approximate total of 1.3 billion requests sent to and processed by the GCP servers. We also collected data for smaller timing differences using millisecond timestamps but leave those out of the reporting because the number of requests needed for a successful attack exceeded our threshold. A collected sample consists of two sets of $1\,000$ baseline requests, where the delay performed on the server is set to only 1 millisecond, and a set of $1\,000$ target requests for which the delay is the same 1 millisecond plus one of the tested delays mentioned above. These delays are set through PHP's `sleep_ns` function which itself also experiences execution time jitter. That is, not every call to the sleep function will take exactly the same amount of time, there will be some variations in runtime, just like a real-world operation would experience. After sending the request, the TCP timestamp values for every received segment are collected. Additionally, we also collect RTTs between the same adversary and victim machines to be able to compare our attack against existing methods.

Our data collection ran for a total of one month on multiple concurrent clients and servers, resulting in several hundreds of gigabytes of network traffic being generated. To collect the data, we incurred a cost of approximately $\$2\,000$ USD for cloud services. The main source for the costs was the egress traffic. Although our servers only returned a small response that fit in a single TCP segment; by sending over a billion requests this still amounted to a high amount of network traffic. Because of the relatively high cost of running the required infrastructure and the time it takes to collect measurements, our work is limited by these practical constraints.

B. Data Analyses

In our analysis, we repeat the attacks 500 times, and assess whether the success-rate over all of those attacks is higher than 95%. To have enough samples for this analysis, and keep the costs of data collection manageable, we reuse 75% of the samples between each subsequent attack iteration. We apply this approach to multiple methods of data analysis:

1) RTT Based Timing Attacks: For the standard timing attacks based on the RTTs we use the box test as defined by Crosby et al., which has become the *de facto* standard method of analyzing RTT based timing attacks [6].

2) TCP Timestamp Based Timing Attacks: For our novel attacks based on the TCP timestamp values we used the `chi2_contingency` test to calculate a unique value for each of the datasets that we then use to find a threshold to differentiate the datasets. This metric was chosen based on the results of prior work that successfully employed this test [21], [20]. Because we found that the `chi2_contingency` test does not perform well when a low number of samples is provided to it, we additionally test each of the datasets against a simple threshold test on the raw data. For the threshold test, we select the smallest value from the datasets and compare those values. Choosing the smallest value over the median or mean is based on the observation from Crosby et al. that for comparing network timings, the lower percentiles work better than higher percentiles or mean values because these distributions are not Gaussian, but rather skewed with a long tail [6]. When using millisecond timestamps, there are only a few discrete bins in which collected values end up, which makes it difficult to calculate the percentiles as used in the box test. With microsecond timestamps enabled, however, we can switch back to the box test. This is possible because there are many more distinct execution time values being measured and calculating percentiles in these datasets is more accurate.

C. Attack Results

1) Millisecond Timestamps: In Table II we show the results of the novel attack (indicated by 'TCPTS') versus the standard timing attack (indicated by 'RTT'). Specifically, this table

[2]Accessible via https://doi.org/10.5281/zenodo.17098889.

Attack	Analysis Method	Coalescing	1 µs	5 µs	10 µs	25 µs	50 µs	75 µs	100 µs	250 µs	500 µs
TCPTS	χ^2 / Threshold	900	–	27 900	900	900	900	900	900	900	900
		100	–	26 100	500	200	100	100	100	100	100
		50	–	32 550	600	200	50	50	50	50	50
		10	–	–	640	230	50	40	20	10	10
		5	–	–	2 320	225	55	35	25	5	5
		2	–	–	2 434	272	48	28	26	12	4
		1	–	–	5 009	305	47	28	25	13	6
RTT	box test	/	–	–	–	10 337	2 944	848	1 089	66	32

TABLE II

RESULTS OF THE ATTACK USING TCP TIMESTAMPS AND A ROUND-TRIP TIME ATTACK. EACH VALUE INDICATES THE NUMBER OF REQUESTS THAT ARE REQUIRED FOR A SUCCESSFUL ATTACK (95% SUCCESS RATE). VALUES MARKED BY A '–' WERE UNSUCCESSFUL WITH UP TO 100 000 REQUESTS.

Attack Analysis Method	Coalescing	500 ns	750 ns	1 µs	5 µs
µs TCPTS box test	900	–	–	–	16 200
	100	–	90 100	45 700	8 500
	50	–	83 000	44 650	7 850
	10	–	80 410	38 370	6 890
	5	–	90 885	40 985	6 245
	2	–	–	45 490	6 654
	1	–	91 349	53 101	5 781

TABLE III

RESULTS OF THE MICROSECOND TIMESTAMPS ATTACK USING THE BOX TEST. EACH VALUE INDICATES THE NUMBER OF REQUESTS THAT ARE REQUIRED FOR A SUCCESSFUL ATTACK (95% SUCCESS RATE). VALUES MARKED BY A '–' WERE UNSUCCESSFUL WITH UP TO 100 000 REQUESTS.

Coalescing	France	Poland	Finland	Combined
900	900	900	900	900
100	100	100	100	100
50	50	50	50	50
10	50	60	50	50
5	50	45	50	50
2	48	48	48	48
1	47	48	48	48

TABLE IV

RESULTS OF THE DISTRIBUTED ATTACK FOR A DIFFERENCE IN RUNTIME OF 50 µs AND DIFFERENT AMOUNT OF COALESCED REQUESTS. THE SEPARATE RESULTS FROM THE THREE CLIENT LOCATIONS ARE ALSO SHOWN. EACH VALUE INDICATES THE NUMBER OF REQUESTS THAT ARE REQUIRED FOR A SUCCESSFUL ATTACK.

shows the number of requests required to be able to successfully differentiate between the baseline and target endpoints where the timing difference between the execution times of those two requests is listed in the header row. For instance, for the timing attack based on the RTT, 2 944 requests are needed to differentiate between a request that took 1 ms to process and one that took 1 ms +50 µs.

It is clear from the data that our novel attack can exploit timing differences that are impossible to differentiate using only RTTs, for delays up to 5 times smaller than those that can be exploited using RTTs. Even when the RTT attack can work, our attacks decrease the required number of requests for a given delay by a factor of over 5 and up to 50.

As mentioned before, the χ^2 test does not work well when only a small number of samples is used. For that reason, we combine the χ^2 test with a simple threshold test and list the outcome of the best performing attack. In practice, we found that the simple threshold test always performed better when fewer than 25 requests were required to distinguish the timing differences. In all other cases the χ^2 test performed better.

2) Microsecond Timestamps: Microsecond timestamps were recently added to Linux as an opt-in feature, and are expected to become more prevalent [50], so we also test our attack using them, with the results in Table III. In contrast to millisecond timestamps, we do not compare to traditional RTT-based attacks, since the RTT-based attack can detect at most 25 µs differences, which is much higher than the highest timing difference tested for the microsecond timing attack.

As shown in Table III, we could detect differences down to 750 ns with an accuracy over 95% using the box test. This timing difference is 33 times smaller than what the best traditional timing attack can detect. Detecting differences of 500 ns is on the verge of practicality: although a 95% success rate was never reached with less than 100 000 requests, when using 100, 50, 10, or 5 coalesced requests, the success rate was around 90%, and when using 2 or 1 coalesced requests, the success rate was around 80%. Similarly, for 750 ns with 2 coalesced requests, the success rate was 90%. Lastly, when using 900 coalesced requests, the success rate of detecting differences of 750 ns and 1 us was roughly 80% and 90%, respectively. We believe the reason for this lower success rate is that the box test is less ideal when relatively few samples are available. That is, when sending up to 100 000 requests using 900 request coalescing, we extract $\lfloor 100000/900 \rfloor = 111$ timestamp differences. We consider it interesting future work to investigate better statistical tests to also achieve 95% success rate for these cases.

3) Distributed Attack: An advantage of our attack is the ability to mount a distributed attack. Multiple attacker clients can collect samples simultaneously from one victim server. This is possible because the measured values are independent of network jitter, so requests from differing network paths will cause similar measurements regardless of their path and the jitter imposed by it. To test the distributed attack, we set up a victim server in Spain and collected data from clients in

France, Poland and Finland simultaneously. We merged the collected data and ran an attack against the resulting dataset. In Table IV, the results of the independent datasets for each of the client locations as well as the results for the combined dataset are shown. When compared to the 50 µs case in Table II and to each other, clearly the distribution of the clients did not have an impact on the results. We limit the distributed attack testing to only one runtime difference because of the previously mentioned cost and time constraints.

D. Success Probability Model

Using our formal model of Section III-B, we can evaluate whether the observed results are in line with what could be expected. As mentioned, the observed value ($S = \lfloor R_0 \rfloor$) is the total execution time taken by the server, rounded down to the nearest accuracy of the TCP timestamp. We perform two comparisons to determine whether it is possible to distinguish between a timing difference of Δ_t, namely whether the two baseline requests are the same: $\lfloor R_{b0} \rfloor = \lfloor R_{b1} \rfloor$, and that the longer execution time results in a higher timestamp: $\lfloor R_{b0} \rfloor < \lfloor R_{b0} + \Delta_t \rfloor$. As we sent requests at random intervals and did not synchronize with the clock tick at the server side, the observed values R_i are randomly distributed in the half-open interval from $\lfloor R_i \rfloor$ until and excluding $\lfloor R_i \rfloor + 1$.

Without considering the jitter of the execution time, we can state that the probability for a successful attack with a single observation is $\mathbb{P}(\lfloor R_{b0} \rfloor < \lfloor R_{b0} + \Delta_t \rfloor)$, or the probability that Δ_t makes the observed result jump to the next tick. The value for this is $\frac{\Delta_t}{1ms}$, where $1ms$ is the accuracy of the timestamp. Similarly, the probability for failure can be expressed as $1 - \frac{\Delta_t}{1ms}$. As such, only a difference of 950µs could be detected with a single request with 95% accuracy. Using n requests, we can determine the probability that all requests fail as $(1 - \frac{\Delta_t}{1ms})^n$, and subsequently the probability for the attack to succeed:

$$1 - (1 - \frac{\Delta_t}{1ms})^n \tag{4}$$

Of course, we need to also take into account the jitter of the execution time on the server. As we make our comparison with three independent requests (two baseline requests, and the target request), the value of the jitter should be counted three times. With J representing the jitter, the success of the attack can then be expressed as:

$$1 - (1 - \frac{\Delta_t}{1ms} + 3J)^n \tag{5}$$

Comparing this probability with our obtained results, e.g. for 250 µs, and an (estimated) jitter value of $J = 1$µs, we find that 11 requests are needed to achieve an accuracy of 95%. This is fairly close to the 13 requests that we found through our empirical results. Similarly, we find that our overall results are in line with the expectation from our formal model, and thus provides more confidence in the validity of these results.

E. Case Studies

1) Attacking TLS: To test our attack against timing leaks in a non-HTTP protocol, we audited open-source TLS libraries for timing attacks [61]. This revealed that the TLS implementation in the Embedded Linux Library (ELL) developed by Intel [62] is vulnerable to the Lucky 13 timing attack. This attack targets CBC-mode encryption, which employs a block cipher where the receiver decrypts the incoming data, removes padding bytes, and then verifies the authenticity of the remaining data by calculating and verifying a Message Authentication Code (MAC). The problem is that the number of padding bytes is determined by unauthenticated decrypted data, specifically the last decrypted byte [61]. As a result, ELL is vulnerable to similar timing attacks as was GnuTLS [61, §6], where the number of input bytes given to the MAC algorithm depend on the last (unauthenticated) decrypted byte, influencing its execution time. These data manipulations cause MAC verification to fail, causing the client to send a TLS Alert. All combined, the value of the last plaintext byte influences how fast the TLS Alert is sent. By trying various modifications to the last value in an encrypted block, and measuring the response time, this byte's value can be recovered [61].

As a proof-of-concept, we implemented an attack against a TLS client that recovers the rightmost byte in an encrypted block. As often done in cryptographic attacks [61], [8], we demonstrate feasibility by distinguishing between two possible plaintext values. When using ELL on a Raspberry Pi 1 B+ with microsecond TCP timestamps enabled, and with the attacker connected to the same Wi-Fi network, we could successfully distinguish between the plaintext bytes 0x00 and 0x01. Note that these two possible plaintext values result in the smallest timing difference, meaning they are the hardest to distinguish.

We also tested the attack against an AWS EC2 t2.micro in North California with microsecond timestamps enabled, with the attacker using a VPS in the United Kingdom. For ethical reasons, we used rate-limiting to avoid overburdening either hosting company. Over this transatlantic connection, our attack successfully distinguished the plaintext bytes 0x00 and 0x41, resulting in the smallest timing difference in the Lucky 13 attack, which in our setup was roughly one microsecond. Moreover, with over 50k samples, we could also distinguish between the bytes 0x00 and 0x01 with a success probability above 50%, corresponding to a timing leak of roughly 150 nanoseconds. Interesting future work is to perform extra measurements to precisely determine the success probability.

The timing leak in ell was assigned CVE-2025-32998.

2) SSH: We also used TCP timestamps to perform a user enumeration timing attack against an OpenSSH server. For this, we built an OpenSSH server that does not include the patch preventing user enumerations [63]. We used a t2.micro server as the victim, where a real user uses sha512 to hash the password, and non-existing users use default bcrypt option.

Like in any realistic scenario, we set up fail2ban and configured it to ban an IP address for one day after 5 login failures [64]. As an adversary, we used multiple AWS t2.micro instances. With this distributed attack setup, we were able

to differentiate (non-)existing usernames with 95% success probability using 30 distributed connection attempts. We also ran the attack while other clients sent 900+ HTTP requests/sec to the targeted server to mimic real-world traffic, which did not noticeably affect the performance of the attack.

This attack is infeasible using a standard RTT-based timing attack, since an adversary would be unable to combine measurements from different sources. We also tested a Raspberry Pi with the real account using sha256, resulting in a timing difference of around 3 ms, which our attack can measure using only a single connection attempt.

3) FTP: Finally, we tested our attack for a user enumeration attack against ProFTPd. We set up a server running ProFTPd on Google Cloud and collected timestamps under two scenarios: (1) without sending any other traffic to the server; (2) while sending approximately 900 requests/sec of synthetic background requests every second to an nginx server running on the same host to mimic real-world background traffic. We found that an attacker only needs a single login attempt to differentiate between existing and non-existing users on the FTP server. Sending about 900 requests/sec did not noticeably change these results.

We also confirmed that FTP allows for request coalescing, meaning an attacker can send multiple login requests at once. Depending on the rate-limiting configuration on the server, the attacker can therefore coalesce requests to FTP and exploit timing attacks with smaller runtime differences.

VI. Discussion

A. Extension for Other Protocols

Any TCP-based protocol can be vulnerable to our attack, as evidenced by our case studies. Whether our optimized coalesce-based method is also feasible depends on whether an attacker can send multiple requests back-to-back and whether or not the server processes those requests sequentially.

1) QUIC as a Transport Protocol: In the newer QUIC protocol, there are no timestamps exposed, rendering the attack impossible. QUIC timestamp frames have been proposed in an Internet Draft, which has been expired since 2023 [65]. If this extension is deployed, it might expose all applications running over QUIC to attacks analogous to our TCP-based ones.

2) High-Accuracy Timestamps: Previously, proposals were drafted to use higher resolution timestamps [45], [48], [49], [46], [47]. As we have shown, there is now also an implementation of microsecond timestamps in the Linux kernel that can be enabled through a route option [50], [51]. We have shown that microsecond timestamps can be used effectively to exploit timing attacks. Therefore, users of this feature should take care not to expose these highly accurate timestamps to the public Internet. We showed that there are only a handful of servers that currently use timestamps more accurate than a millisecond, however, in the future, this number could increase as developers and system administrators find and enable the now optional microsecond timestamps.

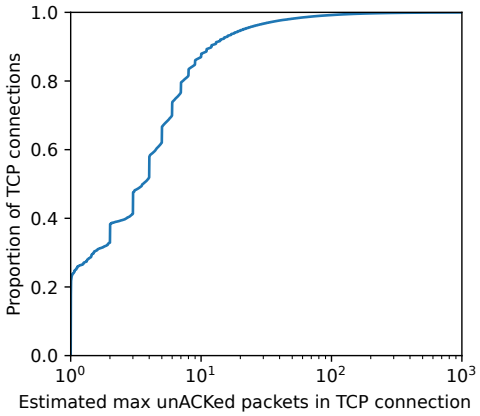

Fig. 4. Cumulative distribution of TCP connections based on their peak number of unacknowledged packets during the connection. This analysis is based on 14 days of 15-minute traffic traces from the WIDE backbone project [68], encompassing over three million real-world TCP connections.

B. Defenses

In addition to well-known defenses such as constant-time implementations, which are known to be difficult to implement in practice, we propose two general defenses:

1) Decreasing Timestamp Frequency: When timestamps are enabled, a timestamp is added to every segment sent by a host. Nishida proposed to decrease this frequency to save bandwidth [66], [67]. In specific cases this can be a viable defense against our attack. We note that when an attacker is able to send hundreds of coalesced requests, it is very likely there will always be at least two response-segments that include a timestamp, thereby making an attack feasible, with the only limitation that the attack cannot choose how many coalesced requests they will include in their measurements. Each of the collected samples may in this case result in a different multiplicative factor of the runtime, which will make attacking more difficult. In the basic attack method, this defense slows the attacker by increasing the number of requests needed, making the attack more time and resource-intensive.

2) Obfuscated Timestamp Values: As a more general and robust defense, we propose to obfuscate timestamps by replacing them with an incremental counter, where the sender maintains a table that maps the counter to timestamp values. We opt for this approach over the idea of encrypting the timestamps because it is compatible with the receiver's PAWS mechanism. While the other side of the connection will not be able to read the actual values of the timestamps anymore, they can still reflect the counter value back to the sender, and the sender can use its table to map it back to the original timestamp value.

To estimate the size of the mapping table, we analyzed the peak number of unacknowledged packets in real-world TCP connections, based on two weeks of network captures from the WIDE backbone that connects universities and research organizations to the Internet in Japan [68], covering slightly

more than 3 million TCP connections. This revealed that a table of 10 entries can handle 87% of TCP connections, and 20 entries can handle 95% (see Figure 4). Here, each entry would take up 8 bytes, namely the counter value and corresponding timestamp. Connections with many unacknowledged packets typically transferred more than 50 MiB of data, and for these connections, or for connections that have no secret-dependent operations, the defense can be disabled. Although this table introduces some overhead, and it might interfere with middleboxes that (try to) use the intercepted timestamps, the advantage is that it is compatible with PAWS.

C. Limitations

While our novel attack improves the limits of timing attacks, there are also limitations that have to be overcome or might prevent the use of the attack against specific hosts.

Primarily, the preconditions required for the attack and discussed in Section IV show specific limitations. Timestamps are optional and they have to be enabled. For the coalescing attack, it must be possible to coalesce requests in specific ways to be able to let them arrive at the server at the same time.

The connection should be persistent and the requests cannot be executed concurrently, which means the use of HTTP/1.1 is required for the attack to work. Although HTTP/2 and QUIC have been published for a while, most servers still implement and listen for HTTP/1.1 for backward compatibility.

D. Future Work

In our case studies, we showed that, apart from HTTP/1.1, our attacks also apply against other TCP-based protocols like TLS, OpenSSH, and FTP. Interesting future work is investigating additional TCP-based protocols for their susceptibility, as well as other non-TCP protocols that also contain a timestamp.

The box test was the best-performing method of analysis among the different analysis methods we tested on the microsecond timestamp dataset, which is in line with previous results of Crosby et al. [6]. For low-granularity data, however, the box test cannot perform well. Interesting future work is evaluating different tests for low-granularity data.

VII. Conclusion

While additional features are often added to protocols for various reasons, we show that these extensions should be handled with care. The values used in the TCP Timestamps option can be abused to exploit timing leaks by reading multiple response segments from a server and deriving an accurate runtime from the timestamp values. We gathered data on runtimes derived from TCP timestamp values on multiple servers for multiple timing differences and use this data to show that it is possible to exploit timing leaks using TCP timestamps. The novel attacks we proposed improve a classical attack by a factor of over 5 and can exploit timing leaks that are 5 times smaller for millisecond timestamps, or even 33 times smaller for the new and optional microsecond timestamps. We have shown that the attack works against multiple TCP-based protocols and not only against HTTP/HTTPS. By means of three case studies we have shown exploits against real-world products work in lab settings with and without background traffic that mimics real-world deployments. A major advantage of the attack is that it is fully distributable, which can speed up attacks and bypass IP-based rate-limiting. We showed that most web servers are likely susceptible to these new attacks and propose two possible defenses, one of which completely prevents attacks.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven, and by the Flemish Research Programme Cybersecurity.

Ethics Considerations

We collect data in a lab-environment that is fully under our own control and as such we do not interfere with any production environments. When we scan (a subset of) the Tranco top 1 million domains, we make sure to either close any opened connection with a `FIN` or `RST` in order to not leave any connections open or resources tied up. On top of that, the amount of traffic we send to each server we connect to is intentionally very low in order to not disturb any production environments. We only sent valid requests, no mis- or malformed data.

In our proof-of-concept attack against the ELL library we used rate-limiting to prevent overloading the used hosting providers. We are also in the process of disclosing this vulnerability: it has meanwhile been assigned a CVE.

References

[1] T. Van Goethem, C. Pöpper, W. Joosen, and M. Vanhoef, "Timeless timing attacks: Exploiting concurrency to leak secrets over remote connections," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1985–2002.

[2] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer, 1996, pp. 104–113.

[3] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, "Understanding network delay changes caused by routing events," *ACM SIGMETRICS performance evaluation review*, vol. 35, no. 1, pp. 73–84, 2007.

[4] D. Brumley and D. Boneh, "Remote timing attacks are practical," *12th USENIX Security Symposium (USENIX Security 03)*, Aug. 2003. [Online]. Available: https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical

[5] A. Bortz and D. Boneh, "Exposing private information by timing web applications," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 621–628.

[6] S. A. Crosby, D. S. Wallach, and R. H. Riedi, "Opportunities and limits of remote timing attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 3, pp. 1–29, 2009.

[7] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 355–371.

[8] N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *2013 IEEE symposium on security and privacy*, 2013.

[9] C. Matte, M. Cunche, F. Rousseau, and M. Vanhoef, "Defeating mac address randomization through timing attacks," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 15–20. [Online]. Available: https://doi.org/10.1145/2939918.2939930

[10] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd," in *IEEE Symposium on Security & Privacy (SP)*. IEEE, 2020.

[11] T. D. Morgan and J. W. Morgan, "Web timing attacks made practical," *Black Hat*, 2015.

[12] T. Van Goethem, W. Joosen, and N. Nikiforakis, "The clock is still ticking: Timing attacks in the modern web," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1382–1393.

[13] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript," in *Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21*. Springer, 2017, pp. 247–267.

[14] M. Golinelli and B. Crispo, "Hidden Web Caches Discovery," in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Sep. 2024. [Online]. Available: https://doi.org/10.1145/3678890.3678931

[15] S. Gast, R. Czerny, J. Juffinger, F. Rauscher, S. Franza, and D. Gruss, "SnailLoad: Exploiting Remote Network Latency Measurements without JavaScript," in *33rd USENIX Security Symposium*, 2024.

[16] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2000, pp. 25–32.

[17] E. Bitsikas, T. Schnitzler, C. Pöpper, and A. Ranganathan, "Freaky leaky {SMS}: Extracting user locations by analyzing {SMS} timings," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 2151–2168.

[18] ——, "Amplifying Threats: The Role of Multi-Sender Coordination in SMS-Timing-Based Location Inference Attacks," in *USENIX WOOT Conference on Offensive Technologies*, ser. WOOT '24. Philadelphia, PA, USA: USENIX Association, Aug. 2024.

[19] T. Schnitzler, K. Kohls, E. Bitsikas, and C. Pöpper, "Hope of delivery: Extracting user locations from mobile instant messengers," *arXiv preprint arXiv:2210.10523*, 2022.

[20] V. Vanderlinden, W. Joosen, and M. Vanhoef, "Can you tell me the time? security implications of the server-timing header," in *Proceedings 2023 Workshop on Measurements, Attacks, and Defenses for the Web. No. March, Internet Society*, 2023.

[21] V. Vanderlinden, T. Van Goethem, and M. Vanhoef, "Time will tell: Exploiting timing leaks using http response headers," in *Computer Security – ESORICS 2023*, G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, Eds. Cham: Springer Nature Switzerland, 2024, pp. 3–22.

[22] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, "On routing asymmetry in the internet," in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, vol. 2. IEEE, 2005, pp. 6–pp.

[23] L. De Vito, S. Rapuano, and L. Tomaciello, "One-way delay measurement: State of the art," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 12, pp. 2742–2750, 2008.

[24] B. Cartwright-Cox, "Splitting the ping," Mar. 2021, accessed: 11 sept 2024. [Online]. Available: https://labs.ripe.net/author/ben_cox/splitting-the-ping/

[25] J. Kettle, "Smashing the state machine: the true potential of web race conditions," Aug. 2023, accessed: 06 sept 2024. [Online]. Available: https://portswigger.net/research/smashing-the-state-machine

[26] S. Schinzel, "An efficient mitigation method for timing side channels on the web," in *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2011, pp. 1–6.

[27] A. Mehta, M. Alzayat, R. De Viti, B. B. Brandenburg, P. Druschel, and D. Garg, "Pacer: Comprehensive network {Side-Channel} mitigation in the cloud," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2819–2838.

[28] D. Borman, R. T. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," RFC 7323, Sep. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7323

[29] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.

[30] L. Zhang, R. T. Braden, and V. Jacobson, "TCP Extension for High-Speed Paths," RFC 1185, Oct. 1990. [Online]. Available: https://www.rfc-editor.org/info/rfc1185

[31] L. Reiner and M. Meyer, "The Eifel Detection Algorithm for TCP," RFC 3522, Apr. 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3522

[32] A. Gurtov and L. Reiner, "The Eifel Response Algorithm for TCP," RFC 4015, Feb. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4015

[33] F. Gont, "Reducing the TIME-WAIT State Using TCP Timestamps," RFC 6191, Apr. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6191

[34] S. Shalunov, G. Hazel, J. Iyengar, and M. Kühlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817, Dec. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6817

[35] A. Kuzmanovic and E. Knightly, "Tcp-lp: low-priority service via end-point congestion control," *IEEE/ACM Transactions on Networking*, vol. 14, no. 4, pp. 739–752, 2006.

[36] B. Veal, K. Li, and D. Lowenthal, "New methods for passive estimation of tcp round-trip times," in *Passive and Active Network Measurement*, C. Dovrolis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 121–134.

[37] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 90–102. [Online]. Available: https://doi.org/10.1145/1644893.1644904

[38] H. Yan, K. Li, S. Watterson, and D. Lowenthal, "Improving passive estimation of tcp round-trip times using tcp timestamps," in *2004 IEEE International Workshop on IP Operations and Management*, 2004, pp. 181–185.

[39] B. McDanel, "TCP Timestamping and Remotely gathering uptime information," Mar. 2001, accessed: 06 sept 2024. [Online]. Available: https://seclists.org/bugtraq/2001/Mar/182

[40] V. Hailperin, "Misusing TCP Timestamps," Mar. 2015, accessed: 06 sept 2024. [Online]. Available: https://www.scip.ch/en/?labs.20150305

[41] E. Bursztein, "TCP Timestamp to count hosts behind NAT," Jan. 2005, accessed: 06 sept 2024. [Online]. Available: http://phrack.org/issues/63/3.html#:~:text=[%20TCP%20Timestamp%20To%20count%20Hosts%20behind%20NAT%20]

[42] G. Wicherski, F. Weingarten, and U. Meyer, "Ip agnostic real-time traffic filtering and host identification using tcp timestamps," in *38th Annual IEEE Conference on Local Computer Networks*, Oct 2013, pp. 647–654.

[43] D. A. Borman, R. T. Braden, and V. Jacobson, "TCP Extensions for High Performance," RFC 1323, May 1992. [Online]. Available: https://www.rfc-editor.org/info/rfc1323

[44] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through tcp timestamps," in *Privacy Enhancing Technologies*, R. Dingledine and P. Syverson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 194–208.

[45] N. Cardwell, Y. Cheng, and E. Dumazet, "TCP Options for Low Latency: Maximum ACK Delay and Microsecond Timestamps," Internet Engineering Task Force, Slides, Nov. 2016. [Online]. Available: https://datatracker.ietf.org/meeting/97/materials/slides-97-tcpm-tcp-options-for-low-latency-00

[46] W. Wang, N. Cardwell, Y. Cheng, and E. Dumazet, "TCP Low Latency Option," Internet Engineering Task Force, Internet-Draft draft-wang-tcpm-low-latency-opt-00, Jun. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-wang-tcpm-low-latency-opt/00/

[47] ——, "TCP Low Latency Option," Internet Engineering Task Force, Slides draft-wang-tcpm-low-latency-opt-00, Jul. 2017. [Online]. Available: https://datatracker.ietf.org/meeting/99/materials/slides-99-tcpm-tcp-low-latency-option-00

[48] K. Y. Yang, N. Cardwell, Y. Cheng, and E. Dumazet, "TCP ETS: Extensible Timestamp Options," Internet Engineering Task Force, Internet-Draft draft-yang-tcpm-ets-00, Nov. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-yang-tcpm-ets/00/

[49] ——, "TCP ETS: Extensible Timestamp Option," Internet Engineering Task Force, Slides draft-yang-tcpm-ets-00, Nov. 2020. [Online]. Available: https://datatracker.ietf.org/meeting/109/materials/slides-109-tcpm-tcp-ets-extensible-timestamps-in-microseconds-and-network-rtt-measurements-00

[50] E. Dumazet, "tcp: add support for usec resolution in tcp ts values," 2023, accessed: 23 April 2025. [Online]. Available: https://github.com/torvalds/linux/commit/614e8316aa4ca

[51] ——, “ip route: add support for tcp usec ts,” 2023, accessed: 23 April 2025. [Online]. Available: https://github.com/iproute2/iproute2/commit/a043bea750026
[52] kernel.org, “The linux kernel source-code,” https://github.com/torvalds/linux/blob/038d61fd642278bab63ee8ef722c50d10ab01e8f/net/ipv4/tcp_ipv4.c#L331, 2025, accessed: 21 August 2025.
[53] nginx contributors, “nginx module ngx_http_core_module directives,” accessed: 06 sept 2024. [Online]. Available: https://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_requests
[54] Apache2 contributors, “Apache core features,” accessed: 06 sept 2024. [Online]. Available: https://httpd.apache.org/docs/2.4/mod/core.html#maxkeepaliverequests
[55] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, “Tranco: A research-oriented top sites ranking hardened against manipulation,” in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, ser. NDSS 2019, Feb. 2019.
[56] W3Techs, “Usage statistics and market shares of operating systems for websites,” accessed: 06 sept 2024. [Online]. Available: https://w3techs.com/technologies/overview/operating_system
[57] Y. Cheng, “tcp: mandate a one-time immediate ACK,” Aug. 2018. [Online]. Available: https://lore.kernel.org/netdev/20180809163812.58365-2-ycheng@google.com/T/#u
[58] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1,” RFC 2616, Jun. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2616
[59] R. Marx, B. Pollard, and C. Böttger, *HTTP*. HTTP Archive, 2024, ch. 18. [Online]. Available: https://almanac.httparchive.org/en/2024/http
[60] T. Vissers, T. Van Goethem, W. Joosen, and N. Nikiforakis, “Maneuvering around clouds: Bypassing cloud-based security providers,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1530–1541. [Online]. Available: https://doi.org/10.1145/2810103.2813633
[61] N. J. Al Fardan and K. G. Paterson, “Lucky thirteen: Breaking the tls and dtls record protocols,” in *2013 IEEE symposium on security and privacy*. IEEE, 2013, pp. 526–540.
[62] Intel, “Embedded linux library,” https://git.kernel.org/pub/scm/libs/ell/ell.git/, 2025, accessed: 28 July 2025.
[63] D. Tucker, “Determine appropriate salt for invalid users.” https://github.com/openssh/openssh-portable/commit/9286875a73b2de7736b5e50692739d314cd8d9dc, 2016, accessed: 2025-07-28.
[64] Arch Linux, “fail2ban,” https://wiki.archlinux.org/title/Fail2ban, 2025, accessed: 2025-07-28.
[65] C. Huitema, “Quic Timestamps For Measuring One-Way Delays,” Internet Engineering Task Force, Internet-Draft draft-huitema-quic-ts-08, Aug. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-huitema-quic-ts/08/
[66] Y. Nishida, “Disabling PAWS When Other Protections Are Available,” Internet Engineering Task Force, Internet-Draft draft-nishida-tcpm-disabling-paws-00, Jun. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-nishida-tcpm-disabling-paws/00/
[67] ——, “Disabling PAWS When Other Protections Are Available,” Internet Engineering Task Force, Slides draft-nishida-tcpm-disabling-paws-00, Jul. 2018. [Online]. Available: https://datatracker.ietf.org/meeting/102/materials/slides-102-tcpm-draft-nishida-tcpm-disabling-paws-00
[68] MAWI Working Group, “MAWI working group traffic archive,” https://mawi.wide.ad.jp/mawi/, 2025, accessed: 2025-07-28.