Poster: Exploiting Timing Side-Channel Leaks in Web Applications that Tell on Themselves

Vik Vanderlinden, Tom Van Goethem, Wouter Joosen and Mathy Vanhoef
imec-DistriNet, KU Leuven
{*firstname.lastname*}*@kuleuven.be*

***Abstract*—The performance of remote timing attacks is highly dependent on the network connection that the attack is executed over, where jitter in both the up- and downstream direction can significantly deteriorate an attack's performance. Traditional timing attacks overcome this problem by obtaining a large number of measurements.**

In this poster, we present a technique to remove the inaccuracies caused by downstream jitter in a remote timing attack, which we expect to reduce the number of measurements required to perform a successful timing attack. Our core idea is to exploit timestamps in HTTP responses, whose values are independent of the downstream jitter. To abuse these timestamps, the adversary synchronizes with the target web server's clock edge, after which the observed timestamps allow the adversary to infer secret information.

We present a method to synchronize with the server's clock and discuss how to compensate for the clock drift between the attacker and target machines. To evaluate the feasibility of our technique, we also investigate the occurrence of timestamps in HTTP responses for the top 10,000 sites according to the Tranco list.

***Index Terms*—Side-channel attacks, timing attacks, web-based attacks, network security**

1. Introduction

The first remote timing attack was performed in 2005 by Brumley and Boneh, who used more than 1.4 million samples to leak a 1024-bit RSA key from a server [1]. Historically, to exploit a remote timing attack an adversary has to obtain many samples to be able to differentiate requests in a statistically significant way. Collecting a high number of samples makes the attack more robust against the jitter imposed by the network. By performing a test like the box test proposed by Crosby, Wallach and Riedi, an attacker can confidently differentiate between operations on a server using a quantifiable metric and leak private data [3].

In order to reduce the need for obtaining such a high amount of samples, the network jitter that is present has to be reduced or removed. By eliminating the dependence on one or both network paths, the jitter can be eliminated from the obtained samples. Previous work showed that both up- and downstream jitter can be removed by coalescing multiple requests into a single TCP segment and looking at the order in which the responses are being returned [5]. However, their technique only works over HTTP/2 and requires that the server uses concurrent processing. We propose a new sequential timing attack that eliminates downstream jitter and can leak sensitive information under less strict prerequisites.

First, the core concept of the attack and the prerequisites will be presented. Second, some necessary optimizations to make the attack feasible will be discussed. Specifically, the clocks of the attacker and target machines have to be synchronized in order to leak information from the target. Adding to the complexity, these clocks will experience a relative drift between them over time, due to the minor inaccuracies in the physical hardware-clocks they use. Because the attack takes a non-negligible amount of time, the relative drift between machines should be compensated for in order to keep the synchronization valid throughout the attack. Finally, the occurrence of timing information on the web is discussed due to its vital importance to a successful attack.

2. Proposed Attack

Consider two requests, one of which includes a secret operation that takes additional processing time (the 'target' request), the other does not (the 'baseline' request). When these two requests are sent to a server at exactly the same moment, the expected outcome would be that the response to the target request is returned after the response to the baseline request because additional processing time has passed. Sending both requests can be timed such that the baseline response is returned before some state change on the server and the target response after the state change. The state change is reflected in the respective responses to both the baseline and target requests. The fact that some state is different in both responses can be used by a malicious actor to leak information. One example of state that constantly changes (increments) and is thus a logical choice for an attack is the current time. If a server reflects information about the current time in its responses, this values continuously changes, by definition, over time and can thus be used to leak private information.

The proposed attack eliminates the downstream jitter by exploiting timing information included in HTTP responses from the target server. Because the response is constructed on the server, the timing information that originated on the server travels over the downstream path unchanged while the jitter imposed on this path has no effect on the contents (among which the timing information) of the HTTP response.

3. Clock Synchronization

The amount of responses that are returned close to a target server's clock edge should be maximized to get the largest potential for differentiating between requests. Bringing the moment the response is sent from the target server close to the target clock edge is exactly the goal of the clock synchronization.

An overview of the clock synchronization process is depicted in fig. 1. The middle bar represents the target server, with clock ticks (where the time on the server rolls over to the subsequent value) indicated by vertical bars. Before the clock synchronization is performed, the client has no knowledge about the timing of the server clock ticks (or edges). As shown at the top of fig. 1, the client may send requests but will not necessarily be close to the clock edges of the target server (it is essentially similar to a random initialization). After synchronization, the goal is to have 50% of the responses be returned before the clock edges and 50% after the clock edges. Practically, the client has to find an offset to delay after its local clock tick that moves the sending of the responses on the server-side just the correct amount of time such that they are being returned around the moment of the clock edge, which is shown at the bottom of the fig. 1.

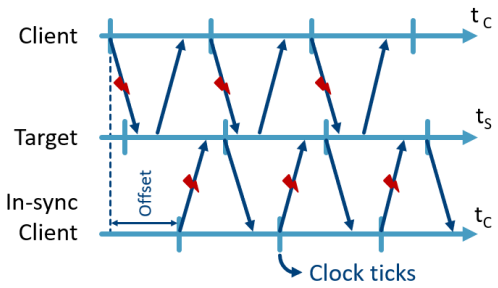


Figure 1. An overview of the effect of clock synchronization. The middle bar represents the target server with clock ticks indicated by vertical bars. The top bar is an unsynchronized client that sends requests to the target server. On the bottom is the same client after the synchronization, now sending requests at an offset in time such that the responses are returned at the exact moments of the target clock ticks. The synchronization process consists of finding the correct offset for the client in such a way that 50% of the responses are returned before and 50% are returned after the target's clock edges.

The synchronization process only uses one request (the 'baseline') that is repeated to perform the synchronization. The attacker machine starts by sending requests over equidistant offsets within an interval (usually one second to start). By observing the responses of the target server, the clock edge may be discovered.

When a response is generated before the clock edge on the target server, consider the time to be t_S. After the clock edge, the time is then incremented to $t_S + 1$. The attacker machine can detect this change and can infer between which offsets the target clock edge occurred. In practice, the detection is less straightforward because the time on the target server is incremented every second and not all requests necessary for the synchronization can be sent within one second. This means that an attacker cannot simply compare the returned timing values directly. Rather, the attacker can use the relative difference between the times on the attacker and target machines, because the time is also incremented on the attacker machine at a more or less similar rate.

When the interval at which the target clock edges occur is found, the attacker can choose to iteratively repeat the process of synchronization within this interval with smaller offsets. This whole process is of course still hindered by the jitter acting upon each request sent over the network. Our preliminary tests show that the synchronization can be performed down to an accuracy of a millisecond with a very low number of required samples. To increase the accuracy of the synchronization further, more optimizations are required, as discussed in the next section.

4. Clock Drift Compensation

Because clocks are inherently inaccurate, be it to a rather small degree that is not disturbing to a human, the clocks of the attacker and target machines may drift away from each other. Relative clock drift has a deteriorating effect on the performed clock synchronization. First because any synchronization that has been successfully performed will only be valid for a small amount of time, until the relative drift will have moved the actual synchronization point away from the detected result. Second, a synchronization of higher resolution may take too such a long time to execute that the actual synchronization point may have already drifted out of focus of the synchronization algorithm (which is iteratively narrowing down on one point) before the synchronization has been completed. Clearly these effects make the attack impossible to perform as is, because an accurate clock synchronization is vital to the success of the attack.

In order to make the attack feasible, the relative drift between the attacker and target machines should be compensated. The absolute drift of a machine's clock can depend on many factors, including but not limited to temperature fluctuations, CPU usage spikes etc. [2]. By using machines in a shared public cloud environment, most factors that may affect the drift are not in control of an adversary. On the other hand, these environments allow an adversary to execute an attack from devices that are physically near the target machines. This means shorter network paths and thus more accurate attacks, thereby incentivizing the use of a public cloud environment.

The relative drift is not stable over time, because it is the sum of two absolute drifts (of the attacker and target machines) which themselves are not stable over time due to the external factors. The relative drift between a machine in our university network in Belgium and a Amazon AWS server in Frankfurt, Germany was monitored for multiple months in 2021. The result depicted in fig. 2 shows that the drift may be approximated as a linear drift most of the time. Note in particular that the time-scale of this figure is large and thus the drift seems extremely large but is actually relatively stable over most periods of a few days. This linear nature gives an adversary the time to approximate the drift close enough to accurately compensate it during the remainder of the attack by taking multiple samples of the target machine clock edge over a few hours and performing a linear interpolation. Most instances at which the drift seems to suddenly change in fig. 2 are reboots of the machine in our university network.

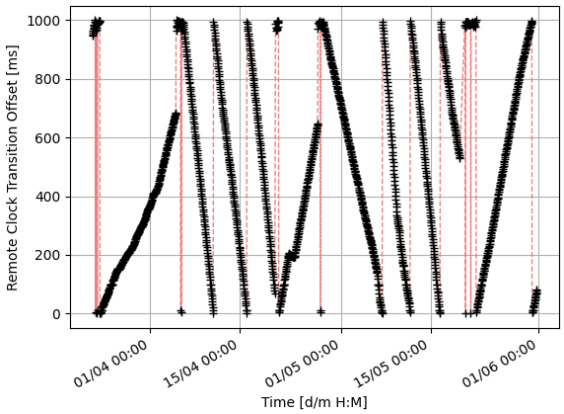


Figure 2. The relative drift between a machine in our university network in Belgium and an instance in the Amazon AWS public cloud environment in Frankfurt, Germany. Sudden changes in drift are mostly reboots of a machine involved in the experiment.

To execute an attack with drift compensation, the drift has to be estimated first, after which all timing values are incremented with the estimated relative drift since the start of the attack (including clock synchronization). Using the drift compensation, we managed to increase the accuracy of the synchronization down to around 10 microseconds. However, it should be noted that this effort significantly increases both the attack time and required number of requests (due to drift compensation and more involved synchronization). An extensive comparison between this method and a standard timing attack regarding the number of requests has yet to be made.

5. Timing Values on the Web

The proposed attack requires the use of timing information embedded in the server's response. This information can either be present in one or more headers or in the body (e.g. embedded within a HTML page) of the HTTP response.

Timing information can occur in different formats. First, there are absolute timing values that represent a specific point in the history or future, usually represented with a year and date. Second, there are relative timing values that represent an offset relative to some absolute time, such as Unix timestamps (the number of seconds since 1 January 1970 UTC). Finally, there are intervals, that define an amount of time and yet are not inherently bound to any absolute time (i.e. an interval such as "5 seconds," is not bound to a specific moment in time).

To detect these different types of timing values, a number of regular expressions have been constructed that match a large number of common formats of timing values. The detection in HTTP headers is a simple string-based match against the each header's value because a HTTP header is essentially a single string. Detection in structured data such as HTML is more difficult. To find timing values in HTML, a bottom-up traversal of the HTML DOM-tree is performed for each visited page.

To evaluate how commonly timing values occur on the web, the Tranco list[1] [4] generated on 22 February 2021 top 10,000 sites were visited by a custom crawler. In total, the crawler successfully visited 41,836 web pages, 5 per site minus some crawler errors. Each page was visited twice with a delay of 10 s between visits. By visiting twice, an easier distinction between variable timing values and static content on a page can be made. It is entirely possible for timing values to occur statically on a site, e.g. the date a blog post was published.

All data gathered was post-processed before any analyses were performed. The post-processing is an important step to filter out static timing values, false positives from the regular expressions and timing values that do not occur consistently on a web page. Because all pages have been visited twice, the post-processor attempts to create a mapping between all found values in both visits and removes those values that are identical on both visits (probably static or false positive results) and that only occur in one out of two visits (no consistent occurrence).

An interesting statistic is that a timing value occurs in the *date* (or *Date*) header in 92.36% of the response documents (of all responses, the document responses with status code 200 are the only ones used for the analyses). Some web pages return a timing value in their *date* header that is unchanged for more than 10 s (the resolution of the timing information is very low). These values are discarded by the post-processing step, being flagged as static values. Even if these values are not entirely static, the granularity of the timing values is too low to use in the attack, thus them being discarded is not an issue.

In total, 5.65% of the documents did not include a single timing value and 37.56% of the responses only have one timing value embedded on them. For most of the responses having one value, this value will be in the *date* header. The median number of timing values per response is 2. The average, however, is 6.39 indicating that there are a number of large outliers. With 990 response including more than 20 timing values and 62 of those even including more than 500 it is clear that these outliers increase the average. Finally, the analyses show that only 22.77% of timing values were found in headers, showing that it is worth including the HTTP response bodies in further analyses.

1. Available at https://tranco-list.eu/list/85NV.

References

[1] D. Brumley and D. Boneh, "Remote timing attacks are practical," in Computer Networks, 48(5), 2005, https://doi.org/10.1016/j.comnet.2005.01.010, pp.701—716.

[2] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in CCS, 2006, https://doi.org/10.1145/1180405.1180410, pp.27—36.

[3] S. A. Crosby, D. S. Wallach and R. H. Riedi, "Opportunities and Limits of Remote Timing Attacks," in ACM Transactions on Information and System Security, 12(3), 2009, https://doi.org/10.1145/1455526.1455530, pp.1—29.

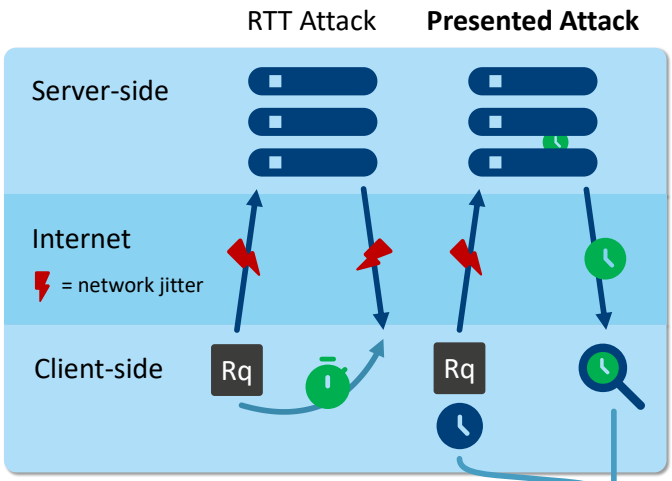
[4] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in NDSS, 2019, https://doi.org/10.14722/ndss.2019.23386

[5] T. Van Goethem, C. Pöpper, W. Joosen and M. Vanhoef, "Timeless timing attacks: Exploiting concurrency to leak secrets over remote connections," in Proceedings of the 29th USENIX Security Symposium, 2020, pp.1985-–2002.

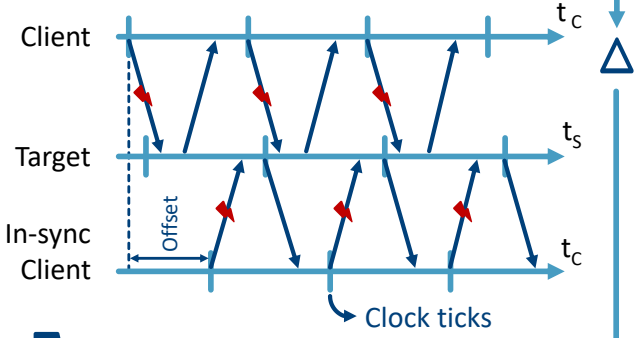
Exploiting Timing Side-Channel Leaks in Web Applications that Tell on Themselves

Vik Vanderlinden[1], Tom Van Goethem[1], Wouter Joosen[1] and Mathy Vanhoef[1]
[1]imec-DistriNet, KU Leuven

Exploiting Timing Values



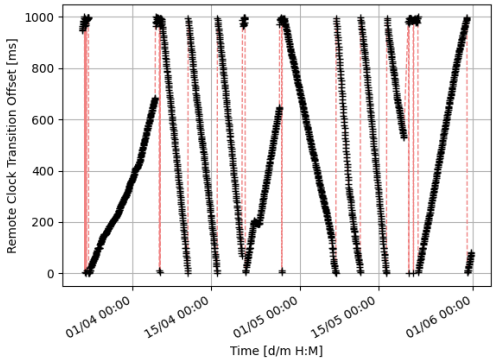
Clock Synchronization



Search offset so 50% before and after

Sync accuracy: **down to 1 ms**

Clock Drift Compensation



Relative drift

Up to **5µs/s** observed

Track, Estimate, Approximate

Increase in Sync accuracy: **down to 10 µs**

Attacking

Estimate Drift | Sync Baseline | Probe Baseline (x2) | Probe Target

Timing Values on the Web

Tranco Top 10k domains
- +-5 pages per domain

Extraction
- Regular expressions
- Full string match on headers
- DOM-tree traversal in HTML

41,836 web pages

Crawl: Visit each page twice with delay

#1 — **Wait 10s** — **#2**

Post-processing

Remove crawl **errors** (10,130)

Attempt to match values from **#1** and **#2**

Remove **unmatched** and **unchanged** values

Remove all except **first-party documents**

24,928 documents

1,569,272 timing values

22,77% in headers
77,23% in bodies

92,36% of *date* headers

Σ	**2**/page median	**6,39**/page on average

0	**> 20**	**> 500**
5,65% of docs	on **990** docs	on **62** docs