Can You Tell Me the Time? Security Implications of the Server-Timing Header

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

Wouter Joosen
imec-DistriNet, KU Leuven
wouter.joosen@kuleuven.be

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

***Abstract*—Performing a remote timing attack typically entails the collection of many timing measurements in order to overcome noise due to network jitter. If an attacker can reduce the amount of jitter in their measurements, they can exploit timing leaks using fewer measurements. To reduce the amount of jitter, an attacker may use timing information that is made available by a server. In this paper, we exploit the use of the *server-timing* header, which was created for performance monitoring and in some cases exposes millisecond accurate information about server-side execution times. We show that the header is increasingly often used, with an uptick in adoption rates in recent months. The websites that use the header often host dynamic content of which the generation time can potentially leak sensitive information. Our new attack techniques, one of which collects the header timing values from an intermediate proxy, improve performance over standard attacks using roundtrip times. Experiments show that, overall, our new attacks (significantly) decrease the number of samples required to exploit timing leaks. The attack is especially effective against geographically distant servers.**

I. Introduction

If a program's execution time depends on a secret, this secret can be leaked by measuring the runtime of the program. This vulnerability is known as a timing side-channel leak. Two and a half decades ago, timing attacks which exploit such timing leaks were documented by Kocher and were focused on the exploitation of cryptographic algorithms with the goal to extract cryptographic keys from the running program [12]. While initially considered impossible, it was later shown that it is in fact possible to execute these attacks remotely, i.e., using a client that is separated from the victim server by a network such as the Internet [6], [5]. The initial remote timing attacks were still focused on cryptographic operations, now happening on the remote server. Meanwhile, several authors found ways to expose sensitive data about users of web applications by utilizing timing attacks [4], [10], [17]. Timing attacks are considered a serious vulnerability and as such, defenses against them are constantly being introduced [13], [1], [3], [2]. As a response to these defenses, that were in some cases considered complete defenses, attack methods were subsequently also improved by researchers, disproving the full effectiveness of the defenses, thereby showing that defending against timing side-channels is far from trivial [15]. These difficulties are worsened by the fact that there are often many ways of exploiting timing leaks. As an example, there have been multiple methods identified that succeed in accurately measuring roundtrip times (the time from sending a request to receiving the subsequent response) in browsers [18], [20], [15]. More recently, a method of excluding noise from used measurements has been presented, which succeeds in significantly improving attack performance under specific preconditions [19].

In this paper we present two attack techniques that, under the right circumstances, can be used to improve the performance of timing attacks. By utilizing timing information that is returned from the server in the *server-timing* header, the measurements used in an attack are more accurate than roundtrip times collected by the client over a possibly noisy network connection. In order to understand the reach of these type of attacks, we first evaluate the use of the header.

Specifically, the following contributions are made:

- We use the HTTPArchive.org datasets to query the current and historic prevalence of the header. We find that while prevalence is still rather low, it has been steadily increasing over time. Additionally, we show that a simple keyword-search in response-headers unveils a large number of sites which expose timing information through a header.
- We present two attack techniques which can be used to improve the performance of a remote timing attack by utilizing the values in the *server-timing* header. We experimentally validate the attack techniques in various settings, which we outline in section IV.
- We propose an update to the W3C Working Draft [22], which specifies the *server-timing* header, to reflect better privacy and security concerns. Currently the working draft has a recommendation that is too lax according to our results.

Coordinated Disclosure: We have disclosed our findings to the authors of the W3C Working Draft and have also contacted Shopify to warn them about the possible security risks of using the *server-timing* header.

II. Background

When timing attacks were introduced two and a half decades ago by Kocher in 1996, most of these attacks were aimed at exploiting cryptographic implementations to extract

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2023
3 March 2023, San Diego, CA, USA
ISBN 1-891562-87-8
https://dx.doi.org/10.14722/madweb.2023.23087
www.ndss-symposium.org

exponents or keys of specific operations [12]. By measuring the execution time of the cryptographic operations it was shown that these attacks were feasible. Not much later, in 2000, the first timing attack in a web browser was shown by Felten et al., who found a method to determine which pages were visited by a user, thus leaking their browsing history [10]. The attacks that are executed in these works always measure local runtimes in a browser or of smartcard-like devices and it was believed that timing attacks against general purpose web servers were infeasible due to the large amount of network noise that packets encounter when traveling over the network to the server [6].

A. Remote Timing Attacks

To show that attacks over a network are in fact practical, Brumley and Boneh exploited cryptographic operations in OpenSSL running on a web server in a local network [6]. Two years later Bortz et al. defined the differences between direct timing attacks and cross-site timing attacks, and at the same time also showed that they could expose private information from a web application over a network [4]. These publications led to a multitude of attack vectors, such as the cross-site search attacks by Gelernter et al., which allow an attacker to test for the existence of search results on another site [11]. More recently, it was shown that arbitrary memory could be read over the network using a timing attack in a remote Spectre attack [16]. Besides practical attacks, Crosby et al. tested multiple statistical methods to compare distributions of measurements and defined the box test to distinguish between roundtrip time distributions, which is now commonly used in timing attacks [9].

Remote timing attacks suffer from an important problem: The packets that are indirectly used to measure the runtime of a program on the backend server have to travel over the network. When packets travel over a network, two important metrics can be defined: the *latency*, or the time it takes a packet to reach the other end of the network, and the *jitter*, also known as the variation on the latency. The network jitter is variable while the network latency is a fixed duration. In theory, the latency can thus be filtered out (i.e., subtracted from the timing measurement), were it not for the fact that each measurement is the sum of the latency and a sample from the jitter distribution, which results in the fact that the exact latency is unknown. Jitter occurs due to the changing load on the target server and due to the middleboxes a packet encounters on its path over the network. These middleboxes can be switches, routers, firewalls, load balancers, or any other networking infrastructure. Each of those middleboxes may handle of high amount of traffic and thus have a queue of incoming packets to handle. When the queue grows in size due to a large amount of ingress traffic, all packets may be delayed for some time while waiting for the middlebox to be able to process all packets in the queue. This waiting process in all the middleboxes adds noise to the time it takes a packet to traverse the network, and it is called jitter [14]. In order to overcome this noise and exploit a remote timing leak, an attacker sends many repeated requests in order to build a distribution of their measurements, which can be evaluated through the use of statistical tests such as the box test [9] to determine whether there is a difference in execution time between two requests.

It is clear that reducing the amount of network jitter in the measurements is an obvious objective for an attacker. The lower the amount of jitter in the measurements, the lower number of measurements the attacker needs to gather in order to successfully exploit a timing leak. Van Goethem et al. succeeded in removing all jitter in their measurements by coalescing multiple requests into one packet, making the server process both packets concurrently, and then looking at the order at which the packets finished processing [19]. Another attack technique is shown by Vanderlinden et al. in which they use the *date* header to synchronize the clocks of the client and the server, after which the amount of responses sent before and after the server clock tick can be used to infer which request has a longer runtime [21]. Using this technique, the downstream jitter is not included in the measurements.

B. The server-timing header

The *server-timing* header is defined in a W3C Working Draft in the Recommendation track [22]. The header is primarily intended for debugging purposes in order to be able to more quickly and easily find performance issues within an application. By adding timing information of specific parts of the execution on the server, performance issues with those specific parts of the application can be easily detected during development. Besides timing information, the header also specifies a means to add textual information, which can be used to indicate cache hits and misses or caching/datacenter locations etc.

The format of the header is defined in such a way that multiple metrics may be communicated through multiple *server-timing* headers. Metrics are separated by commas and each metric has at least a name and optionally a value and/or description. Each metric can consist of multiple parameters, such as the *dur* parameter to communicate durations, or the *desc* parameter to communicate descriptions of the metric. An example use of the header in which three metrics (attack, loc and cache) are included looks like:

server-timing: attack, loc;desc=eu-west
server-timing: cache;desc=write;dur=64

Any timing information added to a duration parameter in a *server-timing* header should have an accuracy of milliseconds, as defined by the 'DOMHighResTimeStamp' type in the working draft [23], [22]. The header is protected by the same-origin policy by default due to the potential sensitive information it includes, however, this can be overwritten by the server if it uses a *timing-allow-origin* or CORS *Access-Control-Expose-Headers* header in its responses, which can be misconfigured [24], [25], [7]. When the server includes certain domains in this header, these domains will be able to access timing information in cross-origin responses.

III. Web Prevalence Scan

In this section, we evaluate the prevalence of the *server-timing* header, the trend of its use over time, and identify some large-scale applications that are using the header. Besides the prevalence of the header, we dive deeper and gain insight into the existence of potentially vulnerable websites that are using the header by checking whether these websites host dynamic

TABLE I. THE ADOPTION OF THE SERVER-TIMING HEADER, CATEGORIZED BY CRUX RANKINGS [8].

rank	# scanned	# headers (%)	# with dur (%)
1 000	695	6.04%	5.47%
10 000	7 304	5.28%	4.08%
100 000	78 750	4.26%	3.34%
1 000 000	838 042	4.61%	4.09%
10 000 000	6 613 812	5.11%	4.12%
100 000 000	10 194 945	5.44%	3.50%

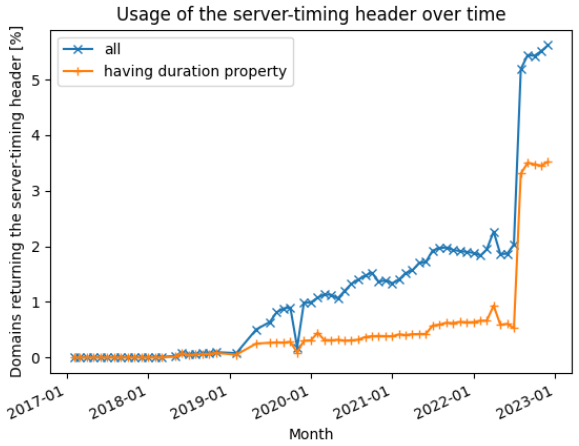

Fig. 1. The percentage of sites using the *server-timing* header in every month between January 2017 and January 2023. The subset of headers including the *dur* parameter is also shown.

content. We also search for sites that expose timing information using other (non-standard) headers that contain timing-related keywords.

A. Adoption

The HTTPArchive.org dataset of September 2022 was queried to check the prevalence of the *server-timing* header. At that moment, 5.44% of sites include the *server-timing* header and 3.50% also have the *dur* parameter set. In absolute numbers, this header is returned from over 550 000 (total) and 355 000 (with *dur* parameter) sites, out of a total of almost 10 200 000 scanned sites.

To gain more insight into the usage of the header on top domains, we used the top-sites ranking of the Chrome UX Report project [8]. In Table I, the adoption categorized by rank is shown. Popular sites use the header slightly more often, but the difference is quite small. The difference in usage of the duration parameter is larger, with top sites have much higher usage of this parameter. A possible reason could be that top sites roll out this header with the duration parameter to evaluate the performance of their products, while less popular sites are using caching mechanisms that add the *server-timing* header to communicate caching-related metrics metrics without duration. Figure 1 shows the header adoption over time. Between 2017 and 2019, the adoption was very low, as can be expected due to the fact that the standard was very new at that moment. Since 2019, there has been a steady increase, with an uptick in the middle of 2022, which occurred due to Shopify, a large e-commerce platform that started using the header and is now

TABLE II. THE MOST USED VALUES OF "SERVER" HEADERS FOR SITES USING THE SERVER-TIMING HEADER.

server	# headers (%)	# with dur (%)
`cloudflare`	3.02%	3.01%
`pepyaka`	1.81%	0.00%
`nginx`	0.27%	0.25%
empty server header	0.14%	0.10%
`apache`	0.07%	0.05%
`cloudfront`	0.02%	0.02%
`microsoft-iis`	0.02%	0.01%
`snow_adc`	0.01%	0.01%
all	5.44%	3.50%

responsible for more than half of the sites using *server-timing* header. The sudden drop in adoption in November of 2019 can be explained due to the lower number of scanned sites in the database for that month.

The rather low adoption at this moment may have several reasons. First, the working draft, as a part of their privacy and security considerations, recommends that the header only be used on authenticated endpoints [22]. The HTTPArchive.org dataset is only queried for homepages and as such the results do not include data on any authenticated endpoints. All of the identified sites that return the header thus go directly against the advise of the specification and add the header on their unauthenticated homepage anyway. Second, the header is primarily used to communicate performance information of the backend server to the front-end. It is possible that many users of the header will therefore only use the header during development or testing.

B. Attribution

Table II shows the most used values for the *server* header (stripped of their specific flavor) for sites using the *server-timing* header. The table shows only *server* values that occur along with a *server-timing* header in more than 0.01% of the scanned sites. The `cloudflare` *server* header occurs most often, which is the case mostly due to Shopify, which uses the header by default while also using some Cloudflare-service(s) that set(s) the *server* header.

Because some popular headers shown in table II are headers set by CDNs (cloudflare, cloudfront), we additionally attempt to find out whether these sites host dynamic or merely completely static content. If a site is completely static, attempting to execute a timing attack does not make sense. Out of the almost 355 000 sites that set the *server-timing* header with the duration parameter, 100 000 were crawled and were checked for the inclusion of forms with a same-origin action. Out of the crawled sites, 1 445 failed to load and 8 598 either do not include a form, or include a form redirecting a user away from the site. Of the remaining sites, 73 717 include at least one form with a POST method. This means that at least 73.12% of these sites have some dynamic behavior. There is of course no guarantee that the dynamic endpoint (the action URL of the form) also exposes the *server-timing* header or that it processes sensitive data.

C. Other headers

We also investigated whether other headers might contain timing information. Querying for this data is non-trivial, be-

TABLE III. OTHER HEADERS POSSIBLY INCLUDING TIMING INFORMATION, OUT OF A TOTAL OF 10 194 945 WEBSITES.

regex pattern	total number of sites	%
`(run)?-?_?time?(ing)?`	894 314	8.8%
`(run)?-?_?time`	341 048	3.3%
`run-?_?time`	195 091	1.9%

cause there is always a trade-off between inclusiveness and false-positive matches. We evaluated three regular expressions against the dataset, each less inclusive than the previous ones. Table III shows the regex patterns used and the corresponding amount of sites including headers that match these regexes. The first regex includes 'runtime', 'time' and 'timing' variants (such as *server-timing*), the last one only 'runtime' variants. Note that there are almost 200 000 sites exposing a runtime header, which sometimes contains data accurate down to microseconds, as we were able to manually verify upon inspection of a few of the resulting headers. However, because the interpretation of the identified headers is not always explicitly defined, we will focus on the *server-timing* header in the remainder of the paper, and consider a closer investigating of custom timing headers interesting future work.

IV. ATTACK TECHNIQUES AND METHODOLOGY

In this section, we introduce the threat models that have been explored, our novel timing attacks that exploit the *server-timing* header values, and we explain the experimental setups that were used to evaluate our novel timing attacks.

A. Threat model

When presenting timing attacks, assumptions about the origin of the attack are often made. A widely accepted assumption is that the attacker is able to rent a virtual machine that is geographically located near the victim machine. This assumption can often not be fulfilled due to the fact that some services still host their web applications on-premises or in private data centers. When exploiting timing leaks against these types of victims, it may be difficult to get accurate results due to the increased network noise in comparison with nearby attackers.

In some cases the attacker might not even have the ability to choose the attacking device, for instance when the attack has to be executed by a piece of malware on some company's employee's computer which executes a timing attack against some victim in the company's private network, to which the attacker usually has no access. In order to execute a cross-site attack (which is executed in a victim's browser) by using additional information included in a response header, the web application under attack has to have a wrong configuration (for instance a wildcard value in an inappropriate location) of their *timing-allow-origin* or CORS *Access-Control-Expose-Headers* headers [24], [25], [7]. This case is not included in our threat model.

Because of this, an attacker may attempt to utilize all additional information available to them in an attempt to exclude as much noise from their samples as possible. The *server-timing* header is one example of additional information that can be used in such a way.

B. Attack techniques

We present our two attack techniques. First, we extend the paradigm of a direct timing attack in such a way that it uses the header information instead of self-collected measurements. Second, by using a proxy that exposes a *server-timing* header, the accuracy of the roundtrip times is increased. In general during a timing attack, an attacker samples two pages: a baseline and a target. The baseline is then used as a reference to compare the target measurements against. The timing attack succeeds if an attacker is able to reliably distinguish the baseline and target measurements.

Improved direct timing attacks: When a server exposes timing information relating to the local runtime, these measurements do not include network jitter. Due to the (significantly) reduced noise in the measurements it will, in theory, be possible to improve a direct timing attack and reduce the required amount of samples to confidently leak private information.

Time-exposing proxy: If a proxy between a client and a server communicates the *server-timing* header in its response, an attacker may use this additional information to their advantage. A logical assumption that can be made is that the connection between a proxy in a production environment and the backend server in said environment is very stable. When compared to the connection between a client on a possibly very noisy network and the proxy, the amount of noise on measurements taken solely by the client versus directly by the proxy may be significant. This effect may be even more significant when the client is located in a different continent than the proxy. In such a case, the client's measurement includes the noisy intercontinental travel as opposed to the proxy whose measurement only includes the local path towards the backend server.

These techniques may show significant improvements over a standard timing attack using roundtrip timings in specific cases. Even more so when attacker-controlled machine is in a geographically distant location.

C. Experimental setup

To evaluate the proposed attacks, we performed tests in controlled settings. Both attack techniques were evaluated from our university's internal cloud environment as well as from a residential network with poor to moderate internet connectivity. By doing this, we are able to compare and see if the improvements of the attacks are greater for networks that initially have bad or noisy characteristics. The data gathered consists of the roundtrip time and the *server-timing* header values. To mimic target endpoints, the page is instructed to wait for a specified amount of time (called the delta between baseline and target endpoints) using PHP's builtin sleep function, before returning the response. For 10 of these timing deltas, a number of samples (usually 50 000) were gathered. In real-world attacks, it can be expected that the server will be under high load or subjected to a high amount of traffic which, depending on the environment, might complicate the attack.

Improved direct timing attack: For the attack where the client accesses the server without a proxy in-between, we set up a server in a cloud-environment in the west of Europe and the east of the USA that exposes the *server-timing* header. The

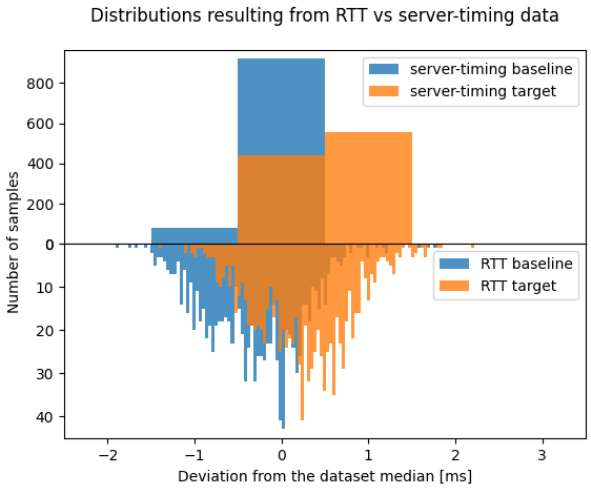

Fig. 2. The difference between a distribution of *server-timing* header values (top) and roundtrip times (RTTs, bottom).

clients in the university or the residential network collect data when sending repeated requests to these servers. For these test-servers, we configured nginx such that it would add the *server-timing* header using nginx's builtin global `$request_time` variable.

Time-exposing proxy: This experiment has more parameters. Again two servers were set up, but now also three proxies were set up. From the clients in Europe, requests were sent to the two EU-based proxies (one proxying to the EU, one to the US) and to the US-based proxy (proxying to the US). When the response from the server returns to the proxies, they are configured to add timing information in their subsequent response. By setting up the experiment like this, we are able to test multiple scenarios using proxies that are close to the server, and that are far away. Even for the proxy on another continent, the hypothesis is that the attack will improve due to the high-bandwidth low-jitter inter-continental connections between cloud provider datacenters. We used nginx servers for both servers and proxies. The proxies are configured to forward requests to the target servers and add a response header with the timing information that is provided as the global variable `$request_time` in nginx.

D. Evaluation

To evaluate the standard roundtrip attacks, the box test was used, since previous work shows that it is the most successful test [9]. For the new attacks using the *server-timing* header, the box test cannot be used. This is because the values of the header have an accuracy of milliseconds, resulting in a distribution that is not granular enough to properly sample percentiles, as shown in the top of Figure 2. To overcome this problem, the samples are compared by means of a statistical test.

The data collected during these attacks is limited to a low number of discrete samples. There are usually no more than a couple of distinct values in the set. A statistical test that can be used as a good metric is one that outputs a value representing the similarity between two datasets. We measure two baseline values and one target value. By doing this, we can validate the timing attack by attempting to distinguish the baseline and target values, but also validate our data by making sure that the two baselines are indistinguishable according to our metric. For all attacks, a threshold can then be searched that correctly distinguishes the baseline vs. target data while still not distinguishing the two baselines. If such a threshold succeeds in correctly classifying more than 95% of the attacks, it is considered valid and the attack is successful.

Pinpointing the exact number of samples required for an attack is a heavy-duty task. To assure our calculations complete within practical time, we do not identify the exact number, but we test a set of 11 sample sizes (5, 10, 20, 50, 100, 200, 500, 1 000, 2 000, 5 000 and 10 000) and test whether the attack would be successful with this number of samples. The results in the next section reflect this decision and should be interpreted as upper bounds on the required number samples.

We evaluated five different statistical tests in the attack, that are all included in the `scipy` python package: $\mathcal{X}^2$, $mathcal X^2$*-contingency*, *Kolmogorov-Smirnov*, *Mann-Whitney U* and *Wilcoxon signed-rank*. These tests output a p-value that can be interpreted as the similarity between the two inputs. Our tests clearly indicate that the Kolmogorov-Smirnov, Mann-Whitney U and Wilcoxon signed-rank tests perform poorly on our datasets as they are often not able to output a p-value that can be used to separate the baseline vs. baseline from the baseline vs. target data. Between the two types of $\mathcal{X}^2$-tests (that both have better performance), the contingency variant has the best performance. The $\mathcal{X}^2$*-contingency* test is implemented as the `chi2_contingency` function in the `scipy` package. Therefore, this test was used in all analyses and is the test used for the results in the section.

V. Results and Discussion

This section presents our experimental results and discusses the implications. First, the results for both attack scenarios are presented and compared to the results of the standard timing attack. After this, the results and their implications are discussed.

A. Results

Table IV shows the results of the improved direct timing attack. Overall, the attack works better than the standard round-trip timing attack, and works both from our university and residential network. The attack performance is quite similar for all attack scenarios and improved the detectable timing difference as can be seen by the fact that all of the attack scenarios are able to distinguish $50\mu s$ in 95% of cases, while this would not be possible using the standard timing attack.

Table V shows the results of the timing attack that uses a proxy. We observe once again a general improvement in attack performance compared to the standard timing attack, where the new attack requires a (significantly) lower number of requests. This is the case both from the university network and from the residential network. These results reflect that, as expected, the attack performance is quite similar across attack scenarios. Note that the results for the proxies in the EU proxying to the US have slightly worse performance, which can obviously be expected. The delta that can be exploited in an attack can

TABLE IV. MINIMUM NUMBER OF SAMPLES REQUIRED TO EXECUTE AN **IMPROVED DIRECT TIMING ATTACK** WITH AN ACCURACY OF AT LEAST 95%. THE UPPER LIMIT IS AT 10 000 SAMPLES. A DASH (-) IS SHOWN IF THE ATTACK WAS UNSUCCESSFUL.

Attack + Test	Network	Server	$5\mu s$	$10\mu s$	$20\mu s$	$50\mu s$	$100\mu s$	$200\mu s$	$500\mu s$	$1ms$	$2ms$	$5ms$
standard RTT box test	university	EU	-	-	-	-	-	-	500	500	50	50
		US	-	-	-	-	-	5 000	2000	-	1 000	100
	residential	EU	-	-	-	-	-	10 000	-	500	100	20
		US	-	-	-	-	-	-	2 000	500	200	20
server-timing $\mathcal{X}^2$*-contingency*	university	EU	-	-	-	10 000	2 000	200	20	10	5	5
		US	-	-	10 000	5 000	1 000	200	20	10	5	5
	residential	EU	-	10 000	-	5 000	1 000	500	50	10	5	5
		US	10 000	-	-	5 000	-	5 000	1 000	-	5	5

TABLE V. MINIMUM NUMBER OF SAMPLES REQUIRED TO EXECUTE A **TIMING ATTACK THROUGH A PROXY** WITH AN ACCURACY OF AT LEAST 95%. THE UPPER LIMIT IS AT 10 000 SAMPLES. A DASH (-) IS SHOWN IF THE ATTACK WAS UNSUCCESSFUL.

Attack + Test	Network	Proxy	$5\mu s$	$10\mu s$	$20\mu s$	$50\mu s$	$100\mu s$	$200\mu s$	$500\mu s$	$1ms$	$2ms$	$5ms$
standard RTT box test	university	EU $\rightarrow$ EU	-	-	-	-	10 000	10 000	500	500	20	20
		EU $\rightarrow$ US	-	-	-	-	10 000	10 000	2 000	200	10	10
		US $\rightarrow$ US	-	-	-	-	-	10 000	2 000	500	5 000	50
	residential	EU $\rightarrow$ EU	-	-	-	-	-	-	5 000	5 000	200	20
		EU $\rightarrow$ US	-	-	-	-	-	-	-	500	200	20
		US $\rightarrow$ US	-	-	-	-	-	-	2 000	5 000	1 000	20
server-timing $\mathcal{X}^2$*-contingency*	university	EU $\rightarrow$ EU	-	-	10 000	10 000	1 000	200	50	10	5	5
		EU $\rightarrow$ US	-	-	-	10 000	5 000	500	100	20	10	5
		US $\rightarrow$ US	-	-	-	5 000	5 000	500	50	10	5	5
	residential	EU $\rightarrow$ EU	-	-	-	-	1 000	-	50	10	5	5
		EU $\rightarrow$ US	-	-	-	-	5 000	500	100	50	10	5
		US $\rightarrow$ US	-	-	-	-	-	10 000	50	10	5	5

be smaller in these new attacks than when using the standard attack, in some cases more than others.

These results have been obtained by using the execution time of the complete request. In some cases more detailed information about requests such as the time to query the database or the time to retrieve records from the cache is exposed in the *server-timing* header. In these cases, the performance of the attack is expected to improve even more.

B. Discussion

In both university network and residential network cases, the *server-timing* header attack improves the attack when compared to the standard attack. One clear factor is that the number of samples required to perform the attack is nearly always lower in our new attack, sometimes drastically. This is very valuable information for an attacker, specifically in combination with the fact that this header is currently often used on sites that also utilize services like Cloudflare, who have strict anti-DDoS protections in place and may block an attacker after a low number of samples have been taken.

The working draft of the *server-timing* header mentions privacy and security considerations where it proposes that responses with potentially sensitive information should only contain timing information if the receiver is authenticated. We argue that this proposal is not strict enough because an attacker can usually authenticate themselves without any problem. This means that if an endpoint exists that can leak sensitive information about *other* users of the application, an attacker can authenticate and use the information included in the *server-timing* header to improve their attack. A second important aspect to note is that the working draft does not mention in any way whatsoever whether this header is meant to only be used during development or also in live production environments.

To mitigate our attacks, the best solution is to not expose the *server-timing* header in production systems, but only enable them during development and/or testing. When the header is still used regardless, the developer should be aware of timing side-channels and protect against them. One pseudo-defense is to decrease the resolution of the timing information. The developer can for example expose values rounded to the nearest multiple of 5 milliseconds. Adding a random delay is another pseudo-defense that can be used in this case. Although no solution is perfect to protect against timing side-channels, these mechanisms can make exploiting the vulnerabilities much more difficult or even impossible.

VI. CONCLUSION

Timing side-channels can be exploited to leak private information about server-state or user profiles. A website may be vulnerable to a remote timing attack when the execution time of backend code is dependent on some private information. An attacker has to collect many samples of the execution time in order to build confidence in their results. By exposing accurate timing information, the attacker can improve their attack and the number of samples that are necessary increases. We investigated the plausibility of using the *server-timing* header to improve an attack.

We have shown that timing information is quite abundant on the web by querying public datasets for the use of the *server-timing* header and other (run)time-related headers. The adoption of the *server-timing* header is steadily increasing and shows a jump in the last year, most probably due to the adoption by Shopify, a large e-commerce platform.

By using the timing values exposed by the *server-timing* header, an attacker can exploit a timing side-channel leak more accurately than before. We presented two attack techniques; One in which the attacker executes a direct attack and is

able to improve the default roundtrip-timing attack due to the addition timing information and a second in which the attacker exploits the fact that a proxy located in a high-bandwidth, low-jitter network adds timing information to responses. These attacks can have benefits when an attacker only has access to a machine in a residential network, when a site is protected against DDoS attacks and blocks large amounts of traffic or when a site is hosted on geographically distant servers.

We suggested that the current W3C Working Draft about the *server-timing* header has security considerations that are not sufficiently strict, we advise to update the document with more detailed recommendations and have contacted the authors of the standard. The ideal defense is to not use the *server-timing* header in production environments. Two pseudo-defenses are rounding the values of the header to for example the nearest multiple of 5 milliseconds or adding a random delay to the values. Both reduce the accuracy of the exposed timing information and thus complicate the attack.

ACKNOWLEDGMENT

This research is partially funded by the Research Fund KU Leuven, and by the Flemish Research Programme Cybersecurity. The authors would also like to thank Tom Van Goethem for his valuable feedback and discussions.

REFERENCES

[1] Alex Christensen, "Reduce resolution of performance.now," https://bugs.webkit.org/show_bug.cgi?id=146531, 2015.

[2] Boris Zbarsky, "Chromium: window.performance.now does not support sub-millisecond precision on windows," https://bugs.chromium.org/p/chromium/issues/detail?id=158234#c110, 2015.

[3] ——, "Clamp the resolution of performance.now() calls to 5us because otherwise we allow various timing attacks that depend on high accuracy timers," https://hg.mozilla.org/integration/mozilla-inbound/rev/48ae8b5e62ab, 2015.

[4] A. Bortz, D. Boneh, and P. Nandy, "Exposing private information by timing web applications," in *16th International World Wide Web Conference, WWW2007*, 2007, pp. 621–628.

[5] B. B. Brumley and N. Tuveri, "Remote Timing Attacks Are Still Practical," in *Lecture Notes in Computer Science*, 2011, vol. 3523, no. II, pp. 355–371. [Online]. Available: http://link.springer.com/10.1007/978-3-642-23822-2_20

[6] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, aug 2005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128605000125

[7] J. Chen, J. Jiang, H. Duan, T. Wan, S. Chen, V. Paxson, and M. Yang, "We still don't have secure cross-domain requests: An empirical study of cors," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18. USA: USENIX Association, 2018, p. 1079–1093.

[8] Chrome Developers, "Chrome ux report," https://developer.chrome.com/docs/crux/, 2022.

[9] S. A. Crosby, D. S. Wallach, and R. H. Riedi, "Opportunities and Limits of Remote Timing Attacks," *ACM Transactions on Information and System Security*, vol. 12, no. 3, pp. 1–29, jan 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1455526.1455530

[10] E. W. Felten and M. A. Schneider, "Timing attacks on Web privacy," in *Proceedings of the 7th ACM conference on Computer and communications security - CCS '00*. New York, New York, USA: ACM Press, 2000, pp. 25–32. [Online]. Available: http://portal.acm.org/citation.cfm?doid=352600.352606

[11] N. Gelernter and A. Herzberg, "Cross-Site Search Attacks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, vol. 2015-Octob. New York, NY, USA: ACM, oct 2015, pp. 1394–1405. [Online]. Available: https://dl.acm.org/doi/10.1145/2810103.2813688

[12] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *CRYPTO - Annual International Cryptology Conference*, 1996, pp. 104–113. [Online]. Available: http://link.springer.com/10.1007/3-540-68697-5_9

[13] A. Mehta, M. Alzayat, R. de Viti, B. B. Brandenburg, P. Druschel, and D. Garg, "Pacer: Network Side-Channel Mitigation in the Cloud," 2019. [Online]. Available: http://arxiv.org/abs/1908.11568

[14] 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, jun 2007. [Online]. Available: https://dl.acm.org/doi/10.1145/1269899.1254891

[15] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10322 LNCS, pp. 247–267. [Online]. Available: http://link.springer.com/10.1007/978-3-319-70972-7_13

[16] M. Schwarz, M. Schwarzl, M. Lipp, and D. Gruss, "NetSpectre: Read Arbitrary Memory over Network," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11735 LNCS, no. July, pp. 279–299, jul 2018. [Online]. Available: http://arxiv.org/abs/1807.10535

[17] M. Smith, C. Disselkoen, S. Narayan, F. Brown, and D. Stefan, "Browser history re:visited," *12th USENIX Workshop on Offensive Technologies, WOOT 2018, co-located with USENIX Security 2018*, no. 1, 2018.

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

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

[20] T. van Goethem, M. Vanhoef, F. Piessens, and W. Joosen, "Request and conquer: Exposing cross-origin resource size," *Proceedings of the 25th USENIX Security Symposium*, pp. 447–462, 2016.

[21] V. Vanderlinden, T. Van Goethem, W. Joosen, and M. Vanhoef, "Poster: Exploiting timing side-channel leaks in web applications that tell on themselves," Genoa, Italy, Jun 2022.

[22] C. Vazac and I. Grigorik, "Server timing: W3c working draft," https://www.w3.org/TR/server-timing/, 2022.

[23] Y. Weiss, I. Grigorik, J. Simonsen, and J. Mann, "High resolution time: The domhighrestimestamp typedef," https://www.w3.org/TR/hr-time-3/#dom-domhighrestimestamp, 2022.

[24] Y. Weiss and N. Rosenthal, "Resource timing: Timing-allow-origin response header," https://www.w3.org/TR/resource-timing/#sec-timing-allow-origin, 2022.

[25] whatwg/fetch contributors, "Fetch standard: Cors protocol," https://fetch.spec.whatwg.org/#http-cors-protocol, 2023.