GPU Application Fingerprinting using Loop-Counting Attacks

Ali Parlakçı, Doğukan Yıldırım, Rana İşlek, Barış Aytekin, Umut Eren Örnek, Damla Sarıçelik, Ali Arda Girgin, Mahdi Ali Pour, and Cemal Yilmaz
Faculty of Engineering and Natural Sciences
Sabanci University
Istanbul, Turkey
{aliparlakci, ydogukan, ranaislek, barisaytekin, umuteren, damlasaricelik, agirgin, mahdialipour, cyilmaz}@sabanciuniv.edu

Abstract—A loop-counting attack increments the value of a counter during a short period of time, repeats the experiments multiple times, and analyzes the sequence of counter values observed, which are affected by the system interrupts, to infer secret information about a victim. These attacks have been so far leveraged for website fingerprinting. In this work, we show that the same attack can also be used for GPU (graphics processing unit) application fingerprinting. More specifically, we demonstrate that the GPU applications typically cause distinguishable patterns of interrupts between the GPU and the CPU, affecting the counter values, such that the sequence of counter values can be analyzed to determine the application running on the GPU. In our experiments conducted on various combinations of operating systems (OS), browsers, media players, and video codecs, we observed that the proposed approach correctly predicted the media player running on the GPU with an average accuracy of 87.99% and the codec of the video being played with an average accuracy of 62.92%.

Index Terms—side-channel attacks, loop-counting attacks, GPU application fingerprinting.

1 Introduction

Side-channel attacks exfiltrate secret information from systems by leveraging information, which is unintentionally leaked from these systems, such as power consumption, execution times, and cache access/miss patterns [1]–[4]. What makes these attacks important and worthy of attention is that, in many of these attacks, neither the algorithms nor the implementations used in the victim systems leak information. However, when these systems share some resources, such as the microarchitectural ones, including CPU and cache memory, by, for exampling, running on the same computing platform with other systems, they suddenly start leaking information.

The loop-counting attack, which has emerged recently [5], is a fine example of side-channel attacks. In this attack, a simple piece of attack code, which can be embedded in an HTML page to be rendered in a browser tab, keeps on updating the value of a counter during a short period of time, repeats the experiments multiple times, and uses the sequence of counter values observed to figure out the website being visited in a different browser tab, i.e., for *website fingerprinting*.

In the loop-counting attack, the primary source of information leakage, thus the culprit of the side channel, is the interrupts, such as the I/O interrupts caused by the arrival or the departure of the network packages being exchanged between the browser and the web server. As the interrupts typically require immediate responses, in the presence of an interrupt, the CPU core is likely to be taken from the malicious code (i.e., the one which repeatedly increments the counter values) and given to the interrupt handler, causing a drop in the counter values observed by the attacker. Indeed, the original paper [5] empirically demonstrates that the sequence of counter values can be used for website fingerprinting as the sequence of I/O operations needed by the websites are likely to differ from one website to another.

In this work, we show that the same attack can also be used for GPU application fingerprinting, i.e., for figuring out the application running on the GPU. More specifically, we demonstrate that the GPU applications typically cause distinguishable patterns of interrupts between the GPU and the CPU, affecting the counter values, such that the sequence of counter values can be analyzed for determining what is actually being processed by a GPU. Note that as GPUs have been extensively used not just for graphics processing, but also for other safety-, mission-, and/or business-critical tasks requiring high performing computing platforms, such as cryptographic operations, artificial intelligence, biomedical analytics, and computational finance [6], [7], [8], [9], [10], [11], [12], they are increasingly becoming targets for cyber attacks. From this perspective, identifying and understanding the side channels exposed by GPUs is, indeed, of practical importance.

In our experiments conducted on various combinations of operating systems (OS), browsers, media players, and video codecs, we observed that the proposed approach correctly predicted the media player running on the GPU with an average accuracy of 87.99% and the codec of the video being played with an average accuracy of 62.92%.

The remainder of the paper is organized as follows: Section 2 presents the threat model; Section 3 introduces the approach; Section 4 presents the empirical results; Section 5 discusses related work; and Section 6 concludes with some

Algorithm 1: Loop-counting attack

```
1  Input: T is the length of attack in seconds and P is
     the length of the increments in milliseconds
2  Output: A sequence of counter values
3  trace ← []
4  attack_timeout ← time()
5  while T ≥ time() − attack_timeout do
6  |  trace_timeout ← time()
7  |  counter ← 0
8  |  while P ≥ time() − trace_timeout do
9  |  |  counter ← counter + 1
10 |  end
11 |  trace ← trace.append(counter)
12 end
13 return trace
```

potential future work ideas.

2 Threat Model

In this work, we use the interrupt-based timing side channels for GPU application fingerprinting by following the co-located attack model. As is the case with the original loop-counting attack [5], the attack code is written in JavaScript running as a service worker in a browser tab, allowing it to be easily embedded in any web page. The victim application, the identity of which is to be exfiltrated, runs on a GPU, which is a part of the same computing platform running the browser. The victim application may have nothing to do with the browser or the tasks carried out by the browser. That is, the victim may be spawn outside the browser and may have no communication with the browser, which is, indeed, the model we used in the experiments. The attack code, on the other hand, relies on the timer provided by the underlying web browser, the resolution of which is often restricted.

3 Approach

In the proposed approach, we, indeed, use the original attack code [13], which implements the main attack loop given in Algorithm 1. The inner loop (lines 8-10) repeatedly updates the value of a counter (line 7) during a period of P milliseconds (in our case, $P = 5$ milliseconds). After the inner loop terminates, the counter value is stored and the outer loop repeats the measurements for a period of T seconds (in our case, $T = 15$ seconds) by resetting the counter before each measurement (lines 5-12). In the presence of interrupts, as the CPU core running the attack code can be allocated for handling the interrupts, the inner loop may iterate fewer times in P milliseconds, which would be reflected on the counter values observed. Consequently, if the GPU applications expose distinguishable temporal patterns of external interrupts, the sequence of counter values can be analyzed to figure out the application running on the GPU.

Note that GPU application fingerprinting is not a completely new idea [14], [15], [16], [17], [18], [19], [20], [21], [22] and [23]. Our work, however, is different in that while the existing approaches depend on some precise information collected about the GPU interrupts, such as their types and frequencies, to which the access from the user space can be restricted, thwarting the attacks, we rely on a simple nested loop doing nothing but incrementing a counter value, which can be embedded and deployed in any web page, without requiring any sophisticated information to be collected about the GPU interrupts.

We, in particular, cast the problem of GPU application fingerprinting as a classification problem. To this end, we use random forest classifiers [24]. The trained model takes as input a sequence of counter values collected during the execution of an application on the GPU and, as output, predicts what the application is. Note that we use the random forest classifiers in this work without claiming that they are the best models to use for the task at hand. After all, our goal in this work is to demonstrate that there is at least one classifier model capable of distinguishing between the GPU applications using the sequences of counter values.

4 Experiments

We carried out a series of experiments to evaluate the proposed approach.

4.1 Subject Applications

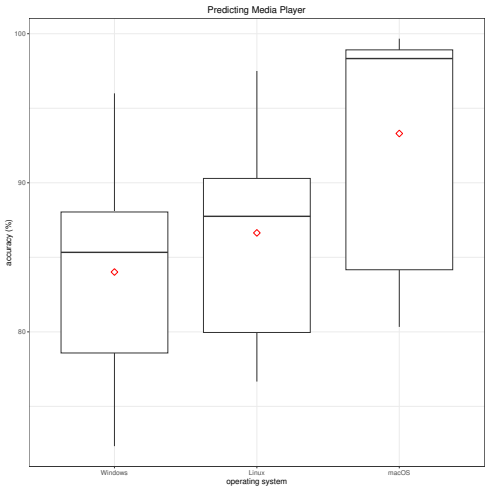
In these experiments, we (without losing the generality) used the proposed approach to predict the video player running on the GPU as well as the codec of the video being played. To this end, we picked 3 media players, namely MPlayer v1.5 (MPlyr), mpv v20230108, and VLC v3.0, together with 4 codecs, namely mp4, 3gp, flv, and mkv, as the subject applications. We chose aforementioned media players because they support multiple operating systems, allowing us to study the sensitivity of the proposed approach to the underlying operating system. Similarly, we chose the aforementioned codecs, because they are well-known codecs, all of which are supported by the selected media players.

We, furthermore, repeated the experiments on different operating systems by using different browsers to run the attack code. More specifically, as the operating systems, we used Windows 10.0.19044, macOS 13, and Linux 5.15, and, as the browsers, we used Chrome 108, Firefox 108, Edge 108, and Safari. The Windows machine was a Lenovo Legion y540 with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 16 GB RAM with the NVIDIA® GeForce® GTX 1660Ti 6GB GDDR6 VRAM, running on Windows 11 (10.0.19044). The Linux machine was a ASUS Zenbook 14" UX431F with an Intel® Core™ i7-8565U Processor 1.8 GHz (8M Cache, up to 4.6 GHz, 4 cores) with the NVIDIA® GeForce® MX150 2GB GDDR5. The macOS machine was a 2021 MacBook Pro 14" with M1 Pro Processor with 8 cores and 16 GB of RAM.

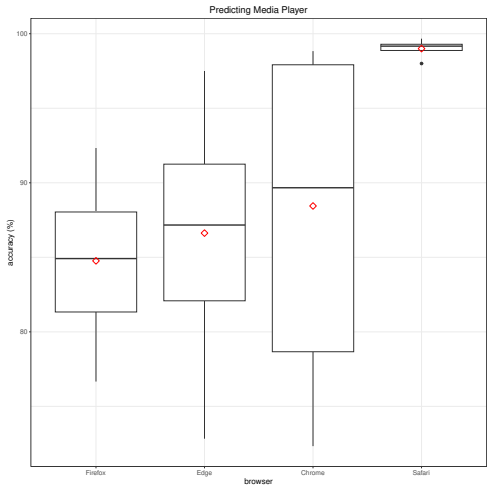
4.2 Operational Model

In the experiments, we used a single video with the different codecs aforementioned in Section 4.1. The attack code was written in JavaScript and executed in a browser tab, using the timer provided by the browser via the `performance.now()` function. The media player of interest was spawned outside the browser in a way, which

Fig. 1: Media player prediction accuracies obtained from (a) different operating systems and (b) different browsers.



(a)



(b)

$!htb$

TABLE 1: The accuracy (A) of predicting the media player (on the left) and the video codec (on the right).

		codec	A
Linux	Chrome	3gp	97.50
		flv	78.17
		mkv	83.33
		mp4	91.67
	Edge	3gp	89.67
		flv	79.83
		mkv	89.00
		mp4	97.50
	Firefox	3gp	86.50
		flv	80.00
		mkv	76.67
		mp4	89.83
macOS	Chrome	3gp	97.67
		flv	98.67
		mkv	98.83
		mp4	98.83
	Firefox	3gp	81.67
		flv	84.50
		mkv	80.33
		mp4	83.17
	Safari	3gp	99.17
		flv	99.17
		mkv	99.67
		mp4	98.00
Windows	Chrome	3gp	72.33
		flv	78.83
		mkv	87.67
		mp4	77.83
	Edge	3gp	82.83
		flv	96.00
		mkv	85.33
		mp4	72.83
	Firefox	3gp	85.33
		flv	89.17
		mkv	87.67
		mp4	92.33

		player	A
Linux	Chrome	MPlyr	80.38
		mpv	81.25
		VLC	93.12
	Edge	MPlyr	74.12
		mpv	67.00
		VLC	92.50
	Firefox	MPlyr	44.50
		mpv	51.00
		VLC	90.00
macOS	Chrome	MPlyr	55.75
		mpv	49.88
		VLC	72.50
	Firefox	MPlyr	39.00
		mpv	27.12
		VLC	30.25
	Safari	MPlyr	47.25
		mpv	51.75
		VLC	73.88
Windows	Chrome	MPlyr	55.75
		mpv	67.50
		VLC	66.62
	Edge	MPlyr	68.88
		mpv	89.75
		VLC	73.50
	Firefox	MPlyr	49.25
		mpv	45.25
		VLC	61.12

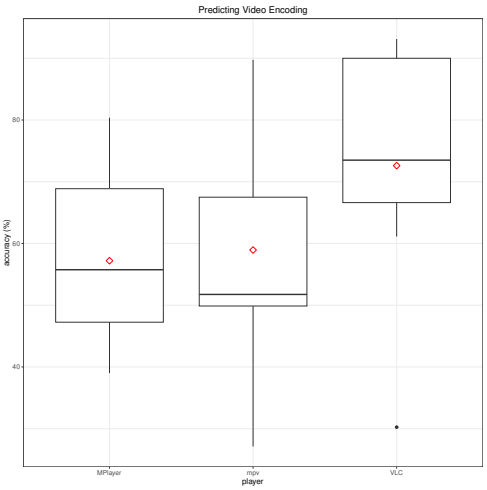


Fig. 2: Predicting the video codec.

guaranteed the use of the GPU. The counter traces were collected during the first 15 seconds of the video with 5 milliseconds increments (i.e., $T = 15$ seconds and $P = 5$ milliseconds in Algorithm 1).

The experiments were repeated 100 times for each valid combination of operating system, browser, media player, and codec. That is, we carried out a total of $100 * 144 = 14400$ experiments. In particular, we couldn't run Safari on the operating systems other than macOS as Safari did not support these operating systems. Furthermore, running the experiments with Edge on macOS almost always failed with no apparent reason. Therefore, we were not able to collect any traces from these configurations.

4.3 Evaluation Framework

For the analysis, we trained and tested random forest classifiers for each experimental setup by using 10-fold cross validation and use the accuracy metric (in percentages) to evaluate the results. The higher the accuracy, the better the predictions are.

Furthermore, given our 4 parameters manipulated in the experiments, namely operating system, browser, media player, and video codec, when we are predicting the media player or the video codec, we trained and tested the models by fixing all the other parameters. Similarly, the aggregated results for a particular scenario were computed by averaging all the available accuracy values obtained in the scenario.

4.4 Data and Analysis

Table 1 summarizes the results we obtained. We first observed that the proposed approach correctly predicted the media player with an average accuracy of 87.99% (+/- 0.02). The maximum detection accuracy, which was 99.67% (+/- 0.01), was obtained for the experimental setup where Safari was running on MacOS while the video was being played in the mkv codec (Table 1).

We then observed that the computing platforms (i.e., the operating systems and the underlying hardware stacks) as well as the browsers used in the experiments leaked different amount of information. Figure 1 presents the results we obtained by visualizing the distributions of the accuracies obtained from the experiments when the independent variables of interest (i.e., the ones reported on the horizontal axes) were fixed.

More specifically, the most vulnerable computing platform was the macOS platform, while the least vulnerable one was the Windows 11 platform (Figure 1a). The average fingerprinting accuracies were 93.31%, 86.64, and 84.01 for the macOS, Linux, and Windows platforms, respectively. Regarding the browsers, the most vulnerable browser was Safari with an average accuracy of 99.00%, whereas the least vulnerable one was Firefox with an average accuracy of 84.76%. The remaining accuracies were 86.63% for Edge and 88.44% for Chrome. Furthermore, it turned out that for the macOS and Safari combination, which suffered from the leakage the most, the operating system was responsible for most of the leakage. In particular, when we consider only the browsers that were able to execute on all of the operating systems, namely Firefox and Chrome, the fingerprinting accuracies were 90.46%, 85.46%, and 83.90%, for the macOS, Linux, and Windows platforms, respectively.

We next observed that predicting the video codec was an harder task than predicting the media player (Table 1b). More specifically, the average prediction accuracy was 62.92%. It turned out that the media players leaked different amount of information (Figure 2). The average accuracies were 72.61%, 58.94%, and 57.21% for VLC, MPlayer, and mpv, respectively.

5 Related Work

Web privacy and security in general and website fingerprinting in particular have been the subject of numerous studies over the past two decades [25]. Hayes and Danezis analyze network traffic for website fingerprinting [26]. Jana and Shmatikov monitor the changes in the browser's memory footprint to determine the website being visited [27]. Kotcher et al. demonstrate that the timing-based side-channel attacks can also be used for sniffing user history and reading text tokens in browsers [28]. Panchenko et al. extend the scope of website fingerprinting to the internet scale [29], while Goethem et al. investigate the prevalence of timing-based side-channels in browsers [30]. Compared to these attacks, the loop-counting attacks leveraged in this work can be considered to be an interrupt-based side-channel attack.

Interrupt-based attacks have also been extensively studied in the literature, highlighting the risks associated with the resulting side channels [15]–[17], [19], [20]. Diao et al. demonstrate an interrupt-based attack on Android devices by analyzing statistical information collected about the interrupts [15]. Similarly, Tang et al. use interrupt statistics to monitor for "sensitive behavior" exposed by the Android apps [19]. Bulck et al. study the microarchitectural timing leaks in the CPU interrupt logic [20]. Mantel and Sudbrock offer various techniques to prevent the covert channels associated with the information leaked from the interrupts and evaluated their efficiency using an information-theoretic approach [17]. Our approach is different than these aforementioned approaches in that, rather than directly using the interrupt statistics, we rely on the effects of the interrupts on the scheduling behavior of the operating systems by leveraging the loop-counting attack.

Interestingly enough, the loop-counting attack also depends on making time measurements. Compared to the other timing attacks, though, this attack, rather than measuring the execution times of fixed tasks, count the number of tasks completed in a fixed amount time. Due to the dependency on time measurements, however, the loop-counting attacks could still be classified as a timing attack. Therefore, many of the approaches for detecting, isolating, and preventing timing attacks at runtime, such as [31], might be used against the loop-counting attacks.

Cemal TODO: Check the refs again.

6 Conclusion

In this paper, we used loop-counting attacks for GPU application fingerprinting. With all the threats to validity in mind, especially the ones concerned with the representativeness of the hardware and software stacks as well as the media players and the video codecs used in the experiments, we believe that our empirical results support our basic hypothesis that loop-counting attacks can also be used for GPU application fingerprinting. We have arrived at this conclusion by noting that the proposed approach correctly predicted the media player running on the GPU with an average accuracy of 87.99% and the codec of the video being played with an average accuracy of 62.92%. One potential avenue for future work for us is to develop approaches for detecting, isolating, and preventing the loop-counting attacks.

References

[1] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007.

[2] J. Szefer, "Survey of microarchitectural side and covert channels, attacks, and defenses," *Journal of Hardware and Systems Security*, vol. 3, no. 3, pp. 219–234, 2019.

[3] J. Betz, D. Westhoff, and G. Müller, "Survey on covert channels in virtual machines and cloud computing," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 6, p. e3134, 2017.

[4] A. C. Atici, C. Yilmaz, and E. Savas, "An approach for isolating the sources of information leakage exploited in cache-based side-channel attacks," in *2013 IEEE Seventh International Conference on Software Security and Reliability Companion*, 2013, pp. 74–83.

[5] J. Cook, J. Drean, J. Behrens, and M. Yan, "There's always a bigger fish: A clarifying analysis of a machine-learning-assisted side-channel attack," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. Association for Computing Machinery, 2022, p. 204–217. [Online]. Available: https://doi.org/10.1145/3470496.3527416

[6] K. Iwai, T. Kurokawa, and N. Nisikawa, "Aes encryption implementation on cuda gpu and its analysis," in *2010 First International Conference on Networking and Computing*. IEEE, 2010, pp. 209–214.

[7] S. A. Manavski, "Cuda compatible gpu as an efficient hardware accelerator for aes cryptography," in *2007 IEEE International Conference on Signal Processing and Communications*. IEEE, 2007, pp. 65–68.

[8] A. E. Cohen and K. K. Parhi, "Gpu accelerated elliptic curve cryptography in gf (2 m)," in *2010 53rd IEEE International Midwest Symposium on Circuits and Systems*. IEEE, 2010, pp. 57–60.

[9] R. Szerwinski and T. Güneysu, "Exploiting the power of gpus for asymmetric cryptography," in *Cryptographic Hardware and Embedded Systems–CHES 2008: 10th International Workshop, Washington, DC, USA, August 10-13, 2008. Proceedings 10*. Springer, 2008, pp. 79–99.

[10] D. Le, J. Chang, X. Gou, A. Zhang, and C. Lu, "Parallel aes algorithm for fast data encryption on gpu," in *2010 2nd international conference on computer engineering and technology*, vol. 6. IEEE, 2010, pp. V6–1.

[11] A. Di Biagio, A. Barenghi, G. Agosta, and G. Pelosi, "Design of a parallel aes for graphics hardware using the cuda framework," in *2009 IEEE international symposium on parallel & distributed processing*. IEEE, 2009, pp. 1–8.

[12] W.-m. W. Hwu, "Recognit." in *GPU Computing Gems Emerald Edition*. Amsterdam, The Netherlands: Elsevier, 2011, pp. 770–778.

[13] J. Cook, "Bigger fish," https://github.com/jackcook/bigger-fish, 2022, accessed: March 24, 2023.

[14] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it:{UI} state inference and novel android attacks," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 1037–1052.

[15] W. Diao, X. Liu, Z. Li, and K. Zhang, "No pardon for the interruption: New inference attacks on android through interrupt timing analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 414–432.

[16] R. Gay, H. Mantel, and H. Sudbrock, "An empirical bandwidth analysis of interrupt-related covert channels," *International Journal of Secure Software Engineering (IJSSE)*, vol. 6, no. 2, pp. 1–22, 2015.

[17] H. Mantel and H. Sudbrock, "Comparing countermeasures against interrupt-related covert channels in an information-theoretic framework," in *20th IEEE Computer Security Foundations Symposium (CSF'07)*. IEEE, 2007, pp. 326–340.

[18] Z. Qian, Z. M. Mao, and Y. Xie, "Collaborative tcp sequence number inference attack: how to crack sequence number under a second," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 593–604.

[19] X. Tang, Y. Lin, D. Wu, and D. Gao, "Towards dynamically monitoring android applications on non-rooted devices in the wild," in *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2018, pp. 212–223.

[20] J. Van Bulck, F. Piessens, and R. Strackx, "Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 178–195.

[21] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems." in *USENIX Security Symposium*, vol. 20, 2009, p. 23.

[22] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, location, disease and more: Inferring your secrets from android public resources," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1017–1028.

[23] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 2139–2153.

[24] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[25] 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.

[26] J. Hayes, G. Danezis *et al.*, "k-fingerprinting: A robust scalable website fingerprinting technique." in *USENIX security symposium*, 2016, pp. 1187–1203.

[27] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 143–157.

[28] R. Kotcher, Y. Pei, P. Jumde, and C. Jackson, "Cross-origin pixel stealing: timing attacks using css filters," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1055–1062.

[29] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.

[30] 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.

[31] A. Javeed, C. Yilmaz, and E. Savas, "Detector+: An approach for detecting, isolating, and preventing timing attacks," *Computers & Security*, vol. 110, 2021.