Dissertation Type: research

DEPARTMENT OF COMPUTER SCIENCE

Timing Attacks in the Modern Web

Ana-Maria Dumitraș

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

18th April 2016

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Ana-Maria Dumitraș, 18th April 2016

Contents

Executive Summary

Supporting Technologies

- HTML
- CSS
- JavaScript
- AngularJS 1
- Bootstrap
- jQuery
- Google Chart Angular
- mathjs
- NodeJS
- Magic ...

Notation and Acronyms

AppCache	:	Application Cache
CPU	:	Central Processing Unit
CSS	:	Cascading Style Sheets
DRAM	:	Dynamic Random Access Memory
HTML	:	HyperText Markup Language
RSA	:	Rivest-Shamir-Adleman Cryptosystem
SCA	:	Side-Channel Attack
SRAM	:	Static Random Access Memory

Acknowledgements

I would like to thank Clifton Suspension Bridge, Sergey Brin and gin.

Chapter 1

Contextual Background

Chapter 2

Technical Background

2.1 Side-Channel Attacks

Timing information, power consumption, electromagnetic emission or heat dissipation measured while the device performs an interesting operation can reveal sensitive information. These leaks based on physical characteristics are known as *Side Channels*. Side-Channel Attacks (SCA) are a powerful set of attacks which rely on recovering secret data by analysing the device rather than exploiting vulnerabilities in the underlying algorithm.

The idea of using Side-Channel information to attack cryptographic schemes was introduced by Kocher in his 1996 paper [1]. Kocher proved that it is possible to find fixed Diffie-Hellman exponents, factor RSA keys and break other cryptosystems by analysing the time it takes to perform certain operations. Side-Channel Attacks try to find the correlation between the side channel information and the internal state of the processing device which is related to the secret parameters involved in the computation.

The literature classifies Side-Channel Attacks by either the level of intrusion (invasive or non-invasive attacks) or the degree of interaction between the adversary and the victim (passive or active attacks).

1. Invasive *vs.* non-invasive: Invasive attacks refer to techniques where the device under attack is permanently modified in order to capture information stored in memory areas or data flowing through the data bus, registers, etc.[2]. Non-invasive attacks only use externally available information such as running time, power consumption, electromagnetic emission or heat emission [3].
2. Active *vs.* passive: Active attacks require the adversary to tamper with the device by either causing the algorithm to execute incorrectly (*Fault Injection*) or modifying any part of the physical implementation. In contrast, a Passive Attack or Side-Channel Analysis will monitor the device's behaviour and record variations of the execution time (*Timing Attack*), the power consumption (*Power Analysis*), the electromagnetic emission (*Electro-magnetic analysis - EMA*) or the heat dissipation.

Side-Channel Attacks are an important class of cryptanalytic techniques. In general, cryptanalysis focuses on studying systems in order to uncover weaknesses that might lead to finding the secret information. Side-Channel Attacks target a specific implementation rather than an abstract algorithm, making them less generic but more powerful [3].

2.1.1 Cache Attacks

Cache attacks are a particular kind of Side-Channel Attacks, which monitor how processes currently running on the device alter the cache. Cache attacks are affected by higher-level security mechanisms, like protection rings, virtual memory, hypervisors and sandboxing. Cache attack are easy to run as they do not require special equipment or excessive computational power. The only requirement is that the attacker is able to run a spy program which shares the cache memory with the victim [4].

2.1.2 Timing Attacks

2.1.3 Oren Attack

2.2 Cache Memory

Cache memory is a small high-speed memory, usually SRAM, located inside the CPU chip. The main role of the cache memory is to speed up the overall runtime of the system by providing fast access to frequently used resources. Cache memory acts as a buffer between the high speed CPU and the slower main memory or DRAM.

A system can only be as fast as its slowest component. Early computers had extremely slow main memory, but CPU were not fast either and their speed was increasing at roughly the same rate. Starting in the 1980s, the CPU clock speed started to increase while memory access time remained relatively slow. Despite having faster CPUs, machines were still slow due to the memory access bottleneck. The increasing gap between CPU speed and memory access time led to the development of the first CPU caches. Nowadays, cache memory is included in every modern CPU from the ultra-low power chips to the highest-end Intel Core i7.

Cache memory increases the overall speed of the system by granting fast access to frequently accessed main memory contents. The algorithm which decides what resources to be stored in cache varies from one architecture to another and is known to change between processor generations [4].

The structure of the cache memory follows a hierarchical scheme. The top level contains the level 1 (L1) cache which is the smallest, fastest and closest to the CPU. The L1 cache is followed by a series of progressively larger and slower memory elements until it reaches the main memory. Typically the cache is split into three levels: L1, L2 and L3. The cache level closest to the RAM is known as last-level cache or LLC [4].

Multi-level caches split into two categories: inclusive and exclusive cache. Inclusive cache requires that lower levels of the cache include all the data which can be found in the upper levels. Exclusive caches guarantee that data can be found in at most one cache level. The two designs have various advantages and disadvantages. Exclusive caches allow more data to be stored while removing data from inclusive caches is generally faster since it only needs to be removed from the last-level cache. Intel's cache architecture is inclusive while AMD's cache architecture is exclusive [4].

When the CPU need to access physical memory it will first look for the respective address in the cache. If the resource does not reside in the cache memory, it will have to be retrieved from the main memory. This is known as a *Cache miss*. A *Cache hit* occurs when the address the CPU is looking for is found in any of the cache memory levels [4].

2.2.1 Cache Organization

Since cache memory is generally smaller than main memory, various techniques are used to map main memory to cache. Main memory is split into cache pages, a cache page is a block of main memory which size is dependent on the cache memory. Each cache page is further divided into cache lines. Based on the mapping mode caches can be split into: direct mapped, n-way set associative and fully associative cache.

- Direct Mapped: Direct Mapped Cache is also known as 1-way set associative cache. In direct mapped mode, the main memory is divided into cache pages equal to the size of the cache. Each cache line can only reside in one position in the cache memory.
- Fully Associative: Fully Associative Cache allows any cache line from the main memory to reside anywhere in the cache memory. In a fully associative scheme, the main memory is only divided into cache lines and pages are completely ignored.
- N-way Set Associative Cache: A Set Associative Cache scheme is a combination of the previous two schemes and the most common technique to map CPU caches. The cache memory is split into

n equal blocks known as cache ways. The main memory is split into cache pages of size equal to the cache way. Each cache way acts like a small direct mapped cache. Each cache line can now be stored in n positions in the cache memory. This decreased the need to overwrite the data in the cache memory.

2.2.2 Intel's Last-Level Cache Structure

Intel's cache architecture is inclusive, so the last level cache contains all the elements stored in the cache. The LLC is divided into slices one for each core of the CPU. Each core is directly connected to a cache slice but also has access to all the other slices [4].

The size of the LLC varies between processor models and ranges between 3MiB and over 20MiB. Due to its relatively large size it is not efficient to iterate through the whole cache when looking for a specific address. The LLC is further divided into cache sets, each block of main memory maps to a specific set in the cache memory. Each cache set contains several cache lines [4].

The Intel Core i7-4960HQ processor is multi-core 4th generation Intel Core processor. The cache memory for this processor has a Smart Cache architecture. Smart Cache is an architecture developed by Intel that allows multi-core processors to share the same last level cache memory instead of having dedicated per-core cache. The processor has 8192 (2^{13}) cache sets, each set being 12-way associative. Every cache set can hold 12 lines of 64 (2^6) bytes each. The total cache size of the Intel Core i7-4960HQ processor is 8192x12x64=6MB [4].

Each cache line can only be stored in 12 positions in the cache memory. The algorithm that determines the mapping between a physical address and an index in cache memory is not public and has changed between processor generations [4]. Hund et el. [5] were able to reverse engineer the mapping in the case of Sandy Bridge processors.

2.3 Web Browsers

A web browser is a software used for interacting with web applications. The main functions of web browsers are retrieving the requested resource and displaying it in the browser window. In order to display the web document, browsers use rendering engines. There are multiple rendering engines available and different browsers use different engines. According to [6] the top 3 most popular browsers among desktop and tablet users are Chrome (56%), Firefox(14%) and Internet Explorer(12%). They all use different rendering engines, Chrome uses Blink, Firefox uses Gecko while IE uses Trident[7].

Rendering engines parse the HTML documents and create an output tree made out of DOM, Document Object Model, nodes. The rendering engine is also responsible for gathering all the style information about the DOM nodes. The final output of the engine is the render tree which is ready to be displayed by the browser[7].

The DOM is a programming interface which allows a more versatile way of accessing the HTML document. It can be easily manipulated by scripting or programming languages in order to change the structure, style or content of the original document[8].

2.4 HTML

HyperText Markup Language, known as HTML, is a markup language used to create web pages. Initially designed as a language for semantically describing scientific documents, HTML became the standard markup language for building web documents. HTML is used to describe the structure and semantic content of a web page but not its functionality[9].

HTML consists of a set of elements, which define the structure of web pages. Most HTML elements are composed of a start tag, `<element>`, and an end tag, `</element>` ,with the content nested in between

the two tags. The end tag is optional and all browsers will display documents which do not contain a matching closing tag for each opened tag; however this technique can produce unexpected results and it is not recommended. Furthermore, strict HTML document validators require that elements have both a start and an end tag[9].

HTML also includes a number of empty elements, elements formed of only a start tag or a start tag with the backslash appended to the element name, `<element/>`, with no content, such as `<br>`, which defines a line break. The HTML standard does not specify which of the two ways of representing an empty element is preferred. Though, the latter provides stricter validation and is accepted by XML parsers[9].

HTML elements can have attributes. Attributes are key, value pairs which determine the behaviour of elements, such as `color`, `font` or `position`. Attributes must be specified in the element's start tag[9].

HTML5 is the 5th version and the current version of the HTML standard. It was released in 2014 by the World Wide Web Consortium (W3C). The current version of HTML introduced a number of new features aimed at simplifying the incorporation of multimedia and graphical content into web based applications[10].

2.4.1 Media Elements

HTML elements are the basic building blocks of web pages. Apart from providing the overall structure of the web page, some HTML elements are used to embed media into a web page. HTML has always provided support for embedding media into web pages, newer versions of the language further simplify the process by introducing elements aimed exactly at this. The `<img>` element, available sine HTML 4.1, allows the insertion of images while the `<audio>` and `<video>` elements introduced in HTML5 enable audio and video content incorporation into web applications.

Image

The `<img>` tag is used to represent an image in a HTML document. The address to the resource to be displayed is stored in the `src` attribute. In order to successfully embed an image in a web page, the `src` attribute must be present and it must contain a valid non-empty URL[10].

The browser starts by downloading the external resource from the address provided in the `src` attribute or an alternative address if this is not available. The `load` event is delayed until the image data has been fetched over the network. Once the image file has finished downloading, the browser will try to display it. If the value of the `src` attribute does not link to a valid image file, the `error` event will be triggered on the `Image` element[10].

Video and Audio

The `<video>` and `<audio>` elements were introduced in HTML5 and enable web developers to easily embed audio and video content into their applications. The most recent version of HTML makes the insertion of a video into a web application as easy as adding an image [10].

Similarly to the `<img>` element, the `Video` and `Audio` elements will first download the external resource and then try to play it. However, these new media elements will not throw an error unless they have finished parsing the entire file[10].

In order to display a media file, the browser will start by downloading the external resource. While the file is being downloaded the `progress` event is being triggered. When the file has been fetched or if the downloading process has been paused the `suspend` event is fired. The browser proceeds by parsing the contents of the file to obtain information such as the length, width, height or extension, etc. of the file. If the media resource is not a valid file the browser will not be able to display it and the `error` event will be triggered on the media element. The time elapsed between the `suspend` and the `error` events is depended on the size of the file. The `error` event fires on the media element when the browser has

finished parsing the file in contrast to the `Image` element, where the event is triggered when the browser learns that the file is not valid, usually as soon as it starts parsing it[10].

2.5 JavaScript

JavaScript is an interpreted scripting language generally used on the client side of web applications. Developed independently by both Netscape and Microsoft, it has been standardized by ECMA International under the name ECMAScript. The most recent version is ECMAScript 2017. ECMAScript and thus JavaScript is supported in all modern browsers. JavaScript is the most widely used client side language [11, 12].

JavaScript's main role is to add dynamic behaviour to web applications through modifying the DOM. The use of technologies to make dynamic and animated web pages is known as Dynamic HyperText Markup, DHTML. DHTML allows scripting languages such as JavaScript to make changes to the DOM in order to alter the appearance or function of the static HTML elements.

To add JavaScript to an HTML document the special `<script>` HTML element is used. JavaScript code can either be embedded in the page or it can be loaded from an external file. Keeping HTML and JavaScript separate is the preferred choice as it helps with both performance and maintenance.

2.6 Offline Experience

Web applications are dependent on network availability and can not function if there is no network connection or in most cases even if the network connection is unstable or not strong enough. Web applications fall into two categories: those that provide access to data: YouTube, Wikipedia or Twitter and those that let users do stuff: Google Keep, Google Docs or CSS Lint.

The first category has access to large amounts of data, but users only use a small portion of it at a given time. One of the ways such applications can be improved is by retrieving the data faster which relies mostly on the network provider and not the application developer. The second category of applications handles a small amount of data and most of the computation happens on the client side. The application data rarely changes and it would be a waste of network resources to retrieve the data every time it is used.

One solution to avoid these needless overuse of the network would be to store some of the data locally. At its simplest an offline web application would have all the needed resources stored locally and when there is no network connection the application would fetch the local copy of the file instead of the remote one. Web developers have started to provide offline access to their applications for some time and there are several methods to achieve this.

The most basic way of storing data is in the browser cache. All browsers are capable of storing web pages if told to do so; however the web developer has no control over the browser cache. It is up to the browser to decide what pages to keep and what pages to remove from the cache when it becomes full.

2.6.1 Application Cache

Application Cache or AppCache is a framework which provides caching of resources in order to be later accessed offline. The AppCache API is part of HTML5 and allows web developers to specify which web pages and resources should be cached by the browser and made available offline.

AppCache provides two ways to cache resources: either by specifying the path to the resource in the cache manifest or by including the cache manifest in the header of the document to be cached. The cache manifest is a text file containing the addresses of all web pages that should be stored locally. Despite being easy to use, AppCache does not allow developers to make any changes to its system.

Although it solves one of the main issues faced by web application developers, offline access, AppCache

has a number of disadvantages. Once a resource is cached it will always be fetched from local storage even if the network condition are good. AppCache does not provide easy update of currently cached resources and is unable to identify any changes made to the remote file. The only way to update the resources cached locally is either to empty the cache or make changes to the manifest file.

All this drawbacks determined developers to move away from AppCache and work towards developing new software for improving users' offline experience. Despite major browsers providing software for interacting with the Application Cache, they are currently removing support this framework.

2.6.2 Web Workers

JavaScript is a single-threaded scripting language. Web applications developed using JavaScript have a main UI thread responsible for DOM access and sequentially running the scripts loaded in the HTML documents. If multiple scripts are loaded on the same page, JavaScript will wait for the previous scripts to finish before starting the next one. Long JavaScript task would make the application unresponsive so this created a need for multi-threading in JavaScript. One way to achieve multi-threaded behaviour in JavaScript is through web workers. Web workers are background JavaScript scripts that run in parallel to the main application. Web workers do not have DOM access; however, they can communicate with each other and the main UI thread.

2.6.3 Service Workers

Application Cache is a great way of providing users with the offline experience. Web application developers have to tailor their product meet the needs of the AppCache mechanism instead of make changes to AppCache to fit the needs of the product. This problem was fixed by Service Workers which allow developers more freedom when determining the offline behaviour of their application.

Service Workers are event driven web workers which sit between the network layer and the application layer. Service Workers were inspired by AppCache, their main purpose is still proving users with the offline experience. What makes Service Workers different is that they give developers complete control over what the offline experience will be.

Compared to AppCache, Service Workers do not store the data needed to run the application in offline mode. Service Workers monitor the network and are able to over-ride default network behaviour. They communicate with other services such as the Cache API that deals with the actual storage part. Typically, a Service Worker would monitor request sent by the application and either serve the data from local storage if available or fetch it over the network and save it locally for later retrieval.

The Cache API stores key/value pairs where the key is the URL and the value is the response returned by fetch. The Cache API ignores the `"no-cache"` and `"no-store"` HTTP header. Resources which contain the `"no-cache"` or `"no-store"` HTTP header can still be cached, however they cannot be displayed.

Service Workers are registered against an origin and a path and requests made from this location will trigger the service worker events. The events will not only fire for every page request within the Service Worker's scope but also for requests made by those pages. Service Workers allow CORS (Cross-Origin Resource Sharing) - requesting a page from a domain outside the domain from which the page originated.

Service Workers run independent of the application they are monitoring. Service Workers need to be linked to the application once it is running and can only monitor applications which started while it was active. The first time the application is started the Service Worker will be linked to the application and start running. Once the Service Worker script is working, the application needs to be restarted, such that the Service Worker can start collecting network information about the application. If the Service Worker is active it can only be closed by stopping the script from the browser, the main application has no control over the Service Worker.

At the moment Service Workers are fully supported in Chrome and Firefox for both desktop and mobile and have basic support in Opera. Microsoft newest browser, Edge, is also working on adding support for this new technology.

2.7 Browser Attacks

Traditionally, web browsers were software applications which allow users to view and interact with content on a web page. The expansion of the Internet determined browser vendors to add new features to their software. The transition from static to dynamic web pages, not only brought richer, more interactive and responsive web applications but also opened the way for a new series of attacks, known as browser attacks. The most popular attacks amongst web applications are by far cross-site scripting (XSS) and SQL injection, over the last few years there has been an increase in attacks of web applications targeting side channel information such as the time it takes to load an external resource.

One of the most popular browser attacks is the disclosure of pages the user has previously visited. The majority of browsers change the color of a link if the user has visited it before. The change can easily be done by using the CSS:`visited` pseudo-class. An attacker can request the computed style using JavaScript and therefore gaining access to the victim's browsers history [13, 14]. Researchers discovered that this techniques was being used by numerous high-profile websites to obtain a user's browsing history [15].

Analysing the style of a web page is not the only way to expose a user's browsing history. A similar attack discovered by Felten et al. [16] uses timing information to determine if a web page has been previously visited by the victim. This attack assumes that recently accessed resources will be cached by the browser in order to provide faster loading times for future visits. By comparing the loading time of different web pages an attacker can determine if a file is in the user's cache, and hence whether or not the user has accessed the file before . Although the attack was discovered over 15 years ago, no one has been able to come up with a solution that will both maintain a browser's caching capabilities and protect the user's privacy [14].

Websites customize their services according to users' locations. For example a user accessing Google from within the United Kingdom will be redirected to www.google.co.uk. Jie et al. have shown that is possible to derive a user's location by simply checking what version of the website they are redirected to [17]. Traditionally a user's location could by determined by checking their IP address; however, the same study shows that this can give imprecise results since users can use VPNs or similar techniques to hide their real IP addresses. Another option, which is more suited for mobile device users, would be to obtain a user's location from GPS sensors. Big websites, such as Google, are able to derive a user's location from their search history. A potential attacker will not have direct access to all the data about the victim but is still able to determine the location by checking where the user is redirected when accessing Google's homepage.

Another class of browser attacks was identified by Bortz et al. in [18]. Their attack allows an adversary to obtain information about the victim's view of another website. Imagine an adversary, Eve, wants to see Alice's view of Bob's Facebook profile page. Eve builds a website, www.eve.com, which when accessed sends a request to Bob's Facebook profile, www.facebook.com/bob. Bob's profile page is private and only visible to Bob's friends. Bob's friends are able to see his photos and wall posts, someone who is not a friend of Bob's will see an empty page. There is a difference in the amount of data received by a user who is friends with Bob and a user who is not. A web application which is able to differentiate between two files of different sizes will also be able to make an assumption about whether or not Alice and Bob are friends. When Alice accesses Eve's malicious application, the website receives Bob's profile information seen from Alice's perspective, approximates the size of the received data and sends the result to Eve. If the size is less than some threshold (say, 500kB), Eve concludes that Alice and Bob are friends; otherwise, she concludes that the two do no know each other.

Bortz. et al. present a series of attacks similar to the example of Alice and Bob. They classify their attacks into two categories: attacks disclosing the value of some boolean variable, such as checking if a user is logged-in or if a email address is a valid user name for a given website, and attacks that estimate the size of hidden data, by forcing a user to access a certain part of a web application and measuring the time it takes to access that resource, an adversary can determine the amount of data the victim has access to.

The attack uses the `<img>` element found in HTML to estimate the size of the data the victim has access to. An attacker loads the address of the target data into an `<img>` element and forces the website to display the resource. The attacker is able to measure the time it takes to download the external resource

which is dependent on the size of the file they are trying to load. Since network conditions are different for each victim, the adversary needs to find a page that will always display the same result for anyone who accesses it, this is usually the 404 page of a web application. By comparing the time it takes to retrieve the target data to the time it takes to access the 404 page, a malicious website could approximate the size of the target data.

The attack presented by Bortz et al. [18] is vulnerable to changing network conditions. Van Goethem et al. expand this attack and focus on detecting vulnerabilities in HTML and JavaScript that give more accurate timing measurements by removing the dependency on a victim's network conditions [14].

2.7.1 Van Attack

Web-based timing attacks have been around for over a decade; however it is very rarely that an attacker is able to obtain meaningful information on the state of the user in a cross-origin website since running the attack requires special network conditions. The increase in popularity of mobile devices means that more and more users connect to the Internet over wireless or mobile network. These types of connections are often unstable and do not meet the necessary requirements to perform a cross-site timing attack. Van Goethem et al. claim to have found new methods to obtain measurements in a cross-site timing attack that are not dependent on network conditions [14].

Van Goethem et al. present several new methods of measuring the load time of an external resources and compares them to existing attacks. Their methods seems to outperform previous attacks in both speed and reliability. They also give several example of how this attacks can be used to obtain information about a user and discuss possible countermeasures [14].

Chapter 3

Project Execution

Chapter 4

Critical Evaluation

Chapter 5

Conclusion

Bibliography

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in CryptologyCRYPTO96*, pp. 104–113, Springer, 1996.

[2] A. Tria and H. Choukri, *Encyclopedia of Cryptography and Security*, ch. Invasive Attacks, pp. 623–629. Boston, MA: Springer US, 2011.

[3] F.-X. Standaert, "Introduction to side-channel attacks," in *Secure Integrated Circuits and Systems*, pp. 27–42, Springer US, 2010.

[4] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox–practical cache attacks in javascript," *arXiv preprint arXiv:1502.07373*, 2015.

[5] R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against kernel space aslr," in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 191–205, IEEE, 2013.

[6] "Stat counter." `http://gs.statcounter.com/`. Accessed: 2016-04-09.

[7] T. Garsiel and P. Irish, "How browsers work: Behind the scenes of modern web browsers." `http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/`. Accessed: 2016-04-09.

[8] "Dom - document object model." `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model`. Accessed: 2016-04-09.

[9] W. W. W. Consortium *et al.*, "Html 4.01 specification," 1999.

[10] R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, E. OConnor, S. Pfeiffer, and I. Hickson, "Html 5.1 specification," *W3C Working Draft*, 2014.

[11] "Clientside scripting languages popularity." `http://w3techs.com/technologies/overview/client_side_language/all`. [Accessed: 2016-04-10].

[12] "Javascript." `https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript`. [Accessed: 2016-04-09].

[13] "Css visited pages disclosure," 2002.

[14] 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*, pp. 1382–1393, ACM, 2015.

[15] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in javascript web applications," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 270–283, ACM, 2010.

[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*, pp. 25–32, ACM, 2000.

[17] Y. Jia, X. Dong, Z. Liang, and P. Saxena, "I know where you've been: Geo-inference attacks via the browser cache," *Internet Computing, IEEE*, vol. 19, no. 1, pp. 44–53, 2015.

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