2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing

Estimating CPU Features by Browser Fingerprinting

Takamichi Saito*, Koki Yasuda†, Takayuki Ishikawa,
Rio Hosoi, Kazushi Takahashi
Department of Computer Science, Meiji University
Kawasaki, Japan
Email: *saito@cs.meiji.ac.jp, †ce66031@meiji.ac.jp

Yongyan Chen
Kunming University of Science and Technology, China

Marcin Zalasiński
Institute of Computational Intelligence, Częstochowa
University of Technology, Poland

***Abstract*—Browser fingerprinting is getting popular in cutting edge Web developers. It typically uses the header information, such as user-agent and plugins. However, the header information can easily be modified or altered by configuration or a browser's extensions. Unlike the header information, hardware information is difficult to be changed, and is regarded as valuable information in browser fingerprinting. Therefore, in this paper, we propose a method that infers the presence or absence of a CPU extension called Advanced Encryption Standard New Instructions (AES-NI) and Intel Turbo Boost Technology (Turbo Boost). Theoretic analysis and experimental results indicate that this method is efficient and feasible for browser fingerprinting.**

Keywords—Browser Fingerprinting; CPU; AES-NI; Turbo Boost

I. INTRODUCTION

With the increasing of application platforms, browsers are becoming more and more complex and computationally intensive recently. Then, the browser has unique features such as its construction or configuration information. Collecting the features make us to identify the browser for web analytics services. Browser fingerprinting is a tracking technology by which web servers can track web visitors and identify its identities. But privacy researchers warn use of browser fingerprinting. Then, because the popular browsers are restricted to collect the information, browser fingerprinting usually collects the information via the browser itself, such as HTTP headers or JavaScript.

While JavaScript with HyperText Markup Language 5 (HTML5) can allow Web developers to design more attractive and functional HTML pages for both web servers and clients.

And, these can make browser fingerprinting be powerful.

However, the most information that can be collected by JavaScript can be changed by configuration settings or browser extensions. According to stability principle that required that fingerprints remain the same over time, we need the number of information that is unlikely to change.

The hardware information tends to remain stable, since it is not easily changed by the user. Hardware information can be prominent fingerprint to track web visitors. Therefore, in this paper, we focus on a CPU feature whose corresponding functions or instructions especially depend on the type of the CPU used. We proposed the method, as new browser fingerprinting, to estimate the existence of Advanced Encryption Standard New Instructions (AES-NI) [1] and the existence of Intel Turbo Boost Technology (Turbo Boost) [2]. We also show the accuracy of the proposed method. In the last, we discuss our methods, and show the countermeasures against our fingerprintings.

II. RELATED WORKS

Mowery et al. [3] constructed strings and images using the Canvas application program interface (API) and Web GL of HTML5. They identified the combinations of GPU, OS, and the browser under the different pixels levels of the drawing results. Using a JavaScript benchmark on a particular browser, they also identified the OS, CPU, and determined the CPU architecture with 45.3% accuracy [4].

Nakibly et al. [5] presented a new fingerprinting method based on the GPU clock rate and clock skew using HTML5 API. Furthermore, they summarized the hardware information accessible from the HTML5 API; namely the GPU, camera, speakers and microphone, motion sensors, and GPS, and introduced battery-powered fingerprinting.

Mulazzani et al. [7] introduced the implementation status of a JavaScript engine as a fingerprint. The fingerprinting method can identify the version or type of browser by using test 262 [15] instead of Sputnik[1] version [8].

Takasu et al. [6] showed that a plurality of hardware information, such as the number of CPU, the presence of GPU rendering, and the presence of media device, can be collected. And they concluded that CPU instructions can be extended to identify the presence or absence of a response for the Streaming SIMD Extensions 2 (SSE2) processor. Therefore, they considered the difference operation result (SSE2). They also argued that the number of CPU cores can be estimated by measuring the difference in operation speed during multithreading by Web Worker API.

III. RELATED TECHNOLOGY

A. AES-NI

AES-NI is an extended instruction equipped in recent Intel or AMD x86 architecture CPU. AES-NI was introduced to the Intel Core processor family in 2010. The six instructions defined in AES-NI provide hardware acceleration for AES

[1] Sputnik is Google's checker to decide whether a Web browser conforms to ECMA-262. It was integrated into the successor test 262 [15].

978-1-5090-0984-8/16 $31.00 © 2016 IEEE
DOI 10.1109/IMIS.2016.108

processing, by improving the speed of AES encryption and decryption of secure data. Four of the instructions support AES encryption and decryption, and the remaining two parts support AES key expansion. In performance analysis implemented in Crypto++ [13], AES/GCM on a Pentium 4 executed cryptographic operations from approximately 28.0 cycles per byte to 3.5 cycles per byte [14]. Therefore, a device mounted with AES-NI can execute cryptographic operations much faster than a non-accelerated device, when AES-NI is available in web browser cryptographic operations. The Web Cryptography API [11] provides AES-NI-based encryption and decryption, but only when the CPU of the web browser's device has AES-NI capabilities. We conclude that it can compute encryption and decryption calculations faster than non-AES CPUs as long as a target CPU supports AES-NI.

Given several current mainstream of the browser, AES-NI performs AES encryption and decryption processing using the Web Cryptography API. Then, we can conduct a series of experiments to estimate existence of AES-NI in a target CPU via browsers.

B. Turbo Boost

Intel Turbo Boost Technology is a function that improves the performance of the CPU by increasing the operating frequency. The recent Intel CPU has the feature. Unfortunately, Turbo Boost does not always work, i.e., raising the frequency of a CPU because its availability depend on a set of factors, such as the number of core, the content of processing and the temperature of the CPU. In addition, the effectiveness of Turbo Boost is reduced in the case of simultaneous running on multiple CPU cores or in the case of high-load processing.

Turbo Boost has evolved through two generations that differ in their implementation: Intel Turbo Boost Technology and Intel Turbo Boost Technology 2.0. In this paper, we only focus on Intel Turbo Boost Technology 2.0 for the proposed method. Hereafter, Intel Turbo Boost Technology 2.0 is denoted as Turbo Boost.

C. SVM

The support vector machine (SVM) is a kind of pattern classifier with supervised learning. Classification by the SVM involves learning labeled data and associating the unknown data with particular labels as a preprocessing for a classification pass. The SVM uses the training data, and the pattern classification model generated from the training data is evaluated on the test data.

In this paper, we use the scikit-learn [9] as the Python library to classify the training samples. Since scikit-learn is provided with a range of pattern classifiers, it prepares a cheat sheet that can select the optimum pattern classifier [10]. Considering the number of training samples, we selected the SVM for classification with a linear kernel in reference as the cheat sheet. Another high-speed SVM library is existed, but as the difference in learning time of the collected number of samples is not observable, we selected the SVC library with significantly higher classification accuracy instead of traditional SVM schemes. Hereafter, the notation SVM strictly refers to an SVC with the linear kernel of scikit-learn.

IV. Proposaled Methods

A. Estimation of AES-NI

For estimating the existence of AES-NI in a target CPU, we presume to measure the difference in operation speed between device with AES-NI and device without AES-NI, with applying Web Cryptography API. The base performance of the operations including cryptographical one depend on not only AES-NI but also CPU's potential performance. Because the processing performance will be different in each CPU regardless of the existence of AES-NI, we cannot simply compare the results. Therefore, in our approach, we compare the ratios of the processing time to the cryptographical operation with and without AES. If the target CPU does not have AES-NI or is disabled, the calculation speed of the referencing arithmetic operation is identical with that of the cryptographical operation of AES. On the contrary, if the target CPU has AES-NI and is enabled, the AES processing time should be faster than the referencing operation that provided by non- cryptographical operations. Therefore, for estimating the existence of AES-NI, we can compare the ratios defined with following formula:

$$\frac{\text{the time of AES operation}}{\text{the time of the referencing arithmetic operation}}$$

We can expect that the ratio of CPU with AES-NI is higher than that of one without AES-NI, and vice versa.

In this paper, the AES operation is executed by Web Cryptography API using AES-CBC encryption which has an encryption key length of 128 bits in the integer array and 64,000,000 decoding iterations. We also calculate PI by the Monte Carlo method as the referencing arithmetic operation for our estimation. We define 50,000,000 times calculation as a chunk. In order to estimate, we compute the time-average of 10 chunks in the consideration of comparability. Finally, we can define the ratio AESrate to compare the targets for estimating as following:

$$AESrate = \frac{\text{the time of AES operation}}{\text{the time of the Monte Carlo calculation}}$$

B. Estimation of Turbo Boost

Unlike the case of estimating AES-NI, it is difficult to observe any instruction in Turbo Boost, or to identify Turbo Boost implements. However, while Turbo Boost can work less effectively under high-load processing, speed of some processing in case of using Turbo Boost are apparently different from in case of processing without Turbo Boost or disable. Therefore, we examined the differences in the processing performance of Turbo Boost using the JavaScript software benchmark Octane 2.0 [12]. Octane 2.0 implemented by JavaScript codes provides processing capacity as a score obtained by measuring 17 benchmark items.

In this paper, we prepared the experimental environment as follows:

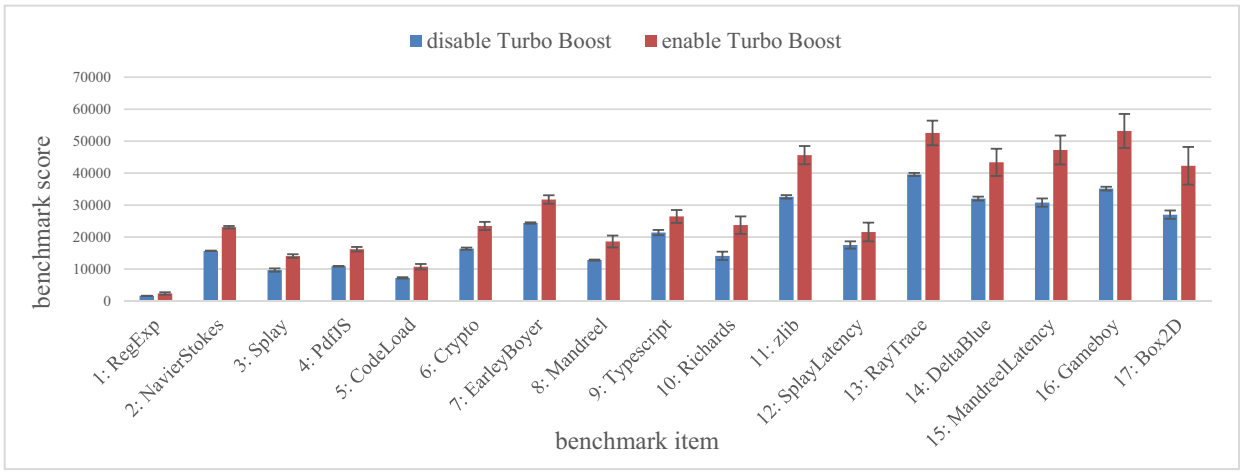

Figure 1. Benchmark scores of Octane 2.0 when Turbo Boost is enabled (orange) and disabled (blue)

OS: Windows 10
CPU: Intel (R) Core (TM) i5-4300U
Memory: 8GB
Browser: Chrome47.0

Figure 1 shows the benchmark scores. We executed the Octane 20 times to obtain standard deviation in cases of able and disabled Turbo Boost, and calculate the mean value of them.

The higher the benchmark score is, the better the performance is. In Figure 1, the benchmark scores are sorted in order of increasing standard deviation. Because the benchmark scores depend on factors other than Turbo Boost such as noise or other program's works, we apply items with small standard deviation to estimate the existence of Turbo Boost.

The average RegExp score, shown in the item 1 in Fig. 1, in disabled Turbo Boost was 1638, while 2353 in enabled Turbo Boost. In the case of average of NavierStokes, the disabled and enabled Turbo Boost shown in the item 15 in Fig. 1, 678.7 and 23,091, respectively. The score ratios of enabled by disabled Turbo Boost were 1.437 in RegExp, and 1.4727 for NavierStokes. Therefore, the effect of Turbo Boost depends on these items. At one time, the ratio NavierStokes/RegExp was 9.572 for disabled Turbo Boost. At the time of the enabling of Turbo Boost, the ratio increased to 9.813. Thus, this ratio can indicate the presence or absence of Turbo Boost.

V. EXPERIMENT

A. Dataset

The target application of our proposed method is web browser fingerprinting. Therefore, in the method evaluation, we collected 350 samples from research participants via browser. In our experiment, we regard 350 samples as our dataset.

In collecting samples, the server obtains the value of user-agent in an HTTP header and the benchmark scores of AES-NI and Octane 2.0. Meanwhile, the server also obtains the CPU model number, which a research participant manually inputs and submits via a HTML form. Moreover, the server spends the elapsed time of collecting all of samples. After the equipment of the CPU, the AES-NI and Turbo Boost has been determined. We confirm the CPU of the model's number by the data sheet of the CPU provided by Intel's site.

The benchmark score is also influenced by browser family or its version, namely the implementation of a JavaScript engine on the browser. Therefore, using the value of user-agent, we divided the samples into four browser categories: Chrome, Firefox, Internet Explorer, and Safari. The numbers of samples in each browser are shown in Table 1. However, the number of sample in Safari was excluded because the sample size was too small for classification by SVM. Thus, we evaluated 341 samples in our experiment.

TABLE I. DISTRIBUTION OF BROWSER

Browser	Number of Samples
Chrome	138
Firefox	111
Internet Explorer	92
Safari	9

B. Experimental Method

In this section, we show the processing time of the proposed method, and evaluate the classification model generated from our dataset.

Although the processing time of the proposed method consists of that of the benchmarking and that of executing SVM, we decide to examine only the processing time of the benchmarking in the browser. The processing time of SVM was negligible to be excluded because it was less than 1ms in this case. Moreover, because the classification of SVM is executed before the estimation process, we exclude the time from fingerprinting.

In this paper, we evaluated the classification model by using leave-one-out cross validation (LOOCV). In LOOCV, while we select a test datum extracted from the samples set, the remaining samples are used as the training data. Then, we repeat the process until all samples have been extracted as the test sample exactly once. Therefore, we create the classification model learned from the training sample, and evaluate its correct classification rate of the test samples.

C. *Estimating of AES-NI*

In the case of estimating the existence of AES-NI, the average time of the benchmarking was 28.15 seconds, accompanied with a standard deviation of 23.24 seconds in 350 samples, although one of the samples required 243.0 seconds.

We calculate the AESrates defined in Section IV in the Chrome, Firefox and Internet Explorer browsers. We show them in Figures 2, 3 and 4, respectively, where samples in the horizontal axis are sorted for a reader's understanding. Namely, the samples of disabled one shown as blue are gathered to the left side in the figure, enabled ones shown as orange are gathered to the right side.

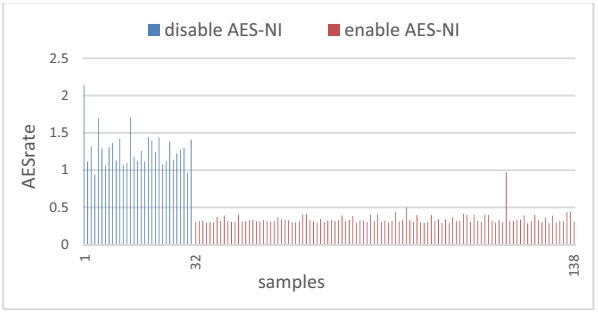

Figure 2. The AESrates in Chrome

In the case of Chrome shown in Figure 2, our proposed method can identify the existence of AES-NI with an accuracy of 99.28%. Therefore, we can conclude that our proposed method estimates the existence of AES-NI in case of Chrome browsers with high accuracy.

Next, in Figure 3, we show the result of AESrates in case of Firefox.

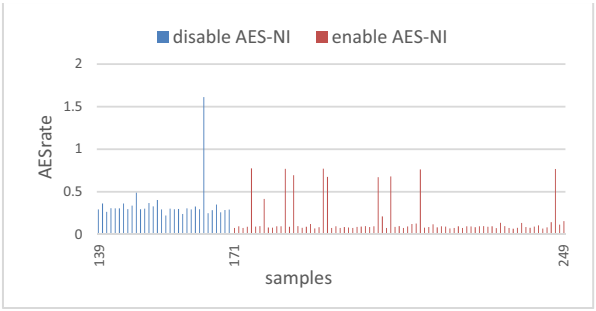

Figure 3. The AESrates in Firefox

In the case of Firefox shown in Figure 3, our proposed method can identify the existence of AES-NI with an accuracy of 71.17%. Different from the result of Chrome, because the results of AESrates of Firefox are unstable, the accuracy of estimation was lower.

Finally, in Figure 4, we show the result of AESrates in case of Internet Explorer.

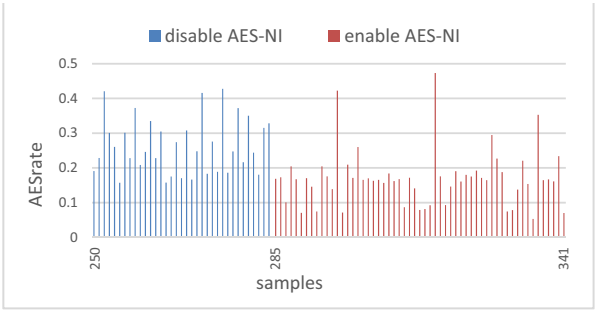

Figure 4. The AESrates in Internet Explorer

In the case of Internet Explorer shown in Figure 4, our proposed method can identify the existence of AES-NI with an accuracy of 77%. The reason why the AESrates were unstable is that the implementation of Internet Explorer cannot bring out the potential performance of a target CPU.

D. *Estimating of Turbo Boost*

If we use the 17 items in Octane 2.0 benchmark on a browser, the average time of the benchmarking was 46.91 seconds, with a standard deviation of 11.41 seconds in 350 samples. One sample required 136.0 seconds as the worst case. Then, in the case of estimating the existence of Turbo Boost, we use only two of the benchmark items, namely RegExp and NavierStokes, to estimate the existence of Turbo Boost.

In the case, we can reduce the processing time to estimate the existence of Turbo Boost by approximately 4 seconds. In this case, we prepared the experimental environment as follows:

OS: Windows 10
CPU: Intel (R) Core (TM) i5-4300U
Memory: 8GB
Browser: Chrome 47.0

Figures 5, 6 and 7 show the NavierStokes/RegExp ratios of the Turbo Boost benchmarks conducted in the Chrome, Firefox and Internet Explorer datasets, respectively. Where, numbers on the horizontal axis indicate the IDs of the samples with disabled (blue) and enabled (orange) Turbo Boost.

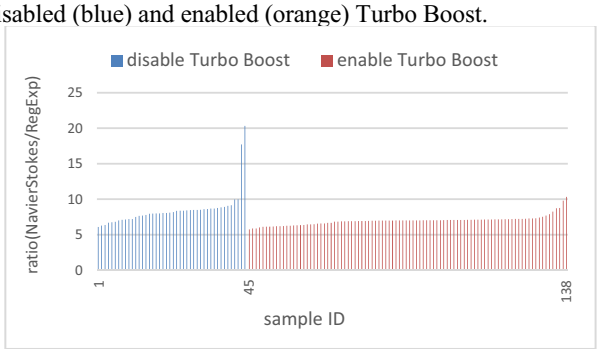

Figure 5. Turbo Boost benchmark measurement result in Chrome

The accuracy of estimations in the Chrome, Firefox and Internet Explorer browsers were 84.78%, 82.88% and 55%, respectively.

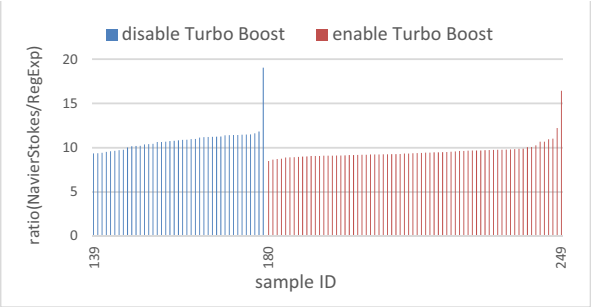

Figure 6. Turbo Boost benchmark measurement result in Firefox

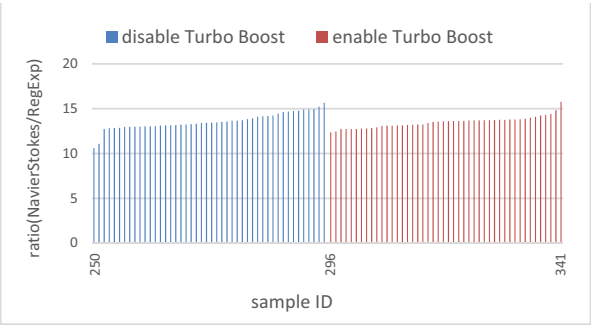

Figure 7. Turbo Boost benchmark measurement result in Internet Explorer

The Turbo Boost benchmark scores were relatively stable in Chrome and Firefox, accompanied with a high accuracy of estimations, but not stable in case of Internet Explorer. Therefore, the proposed method cannot reliably estimate the status of Turbo Boost in Internet Explorer.

VI. Discussion

A. Proposaled Methods

The estimation accuracy of the proposed method was available even in the cross-browsers. Both AES-NI and Turbo Boost statuses, i.e., enable or disable, were estimated with high accuracy in short processing time, in the Chrome browser. Moreover, the benchmark scores were more stable and discriminating in Chrome than in the other browsers. This suggests that Chrome has efficient implementation of the CPU extensions. The estimates were relatively stable in Firefox, but were degraded in Internet Explorer. We conclude that the benchmark scores are destabilized, and that the CPU functions are ineffectively, in Internet Explorer. Therefore, the selection of the browsers affects the estimation accuracy of the proposed method and the operation efficiency of the CPU extensions.

Especially in case of running a high-load processing in the target device, some noises are generated in the benchmark processing. To reduce the noise effects in estimating the existence of Turbo Boost, we selected benchmark scores with low noise effects in the model evaluation. Furthermore, the ratio between the two items, namely RegExp and NavierStokes, realizes an estimation independent of the device or the situation that a high-load processing runs. In future work, we will reduce the noise to negligible levels in the estimation.

The proposed method is rather slow, especially in case of the AES-NI estimation required 28.15 seconds on average. However, the estimating time can be reduced to the demanded requirements of accuracy. For example, the estimation accuracy in the Chrome browser exceeded 99% with 28.15 seconds. If such high accuracy is not required, it can be slightly reduced to shorten the estimation time.

B. Countermeasure to the Proposed Methods

One of countermeasures against the proposed method is to degrade the accuracy of the time-measurement function built in JavaScript. Time measurements in JavaScript are performed by an in-built object called Date and High Resolution Time API. Currently, the Date and High Resolution functionalities measure time with millisecond and micro-second accuracy, respectively. As estimation of Turbo Boost uses both measurements, accurate estimations are precluded with the accuracy decreasing. As one of countermeasures, we introduce the lower-accuracy Date functionality. However, rewriting the JavaScript standard built-in object will likely affect other content on the website.

In our experiment, the proposed method identifies the type of browser, and then classifies the benchmark scores. W use user-agent to obtain name of the target browser. Namely, identifying the target browser is important. It seems possible to disturb the user-agent such like spoofing browser extensions. But we have other methods, e.g., Mowery et al. [4], to identify browser implementation.

VII. Conclusion

In this paper, as one of browser fingerprinting method, we proposed to estimate the existences of AES-NI and Turbo Boost in the target device using the hardware information. We conducted that our method can estimate the existences with high accuracy, i.e., 99.28% for AES-NI and 84.7% for Turbo Boost. Since our method uses Web Cryptography API and the Octane 2.0 benchmark score, it can be applied to the major browser: Chrome, Firefox, Internet Explorer, and Safari.

In case of Chrome, we obtained the highest estimation accuracy. It indicates Chrome is the best implementation among them. On the other, the estimation accuracies of both AES-NI and Turbo Boost were low in Internet Explorer.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 26330162. We are deeply grateful to Y. Iso and N. Kiryu for this work.

References

[1] Intel Advanced Encryption Standard(AES) New Instructions Set https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf

[2] Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors: http://files.shareholder.com/downloads/INTC/0x0x348508/C9259E98-BE06-42C8-A433-E28F64CB8EF2/TurboBoostWhitePaper.pdf

[3] Mowery, K., Shacham, H.: Pixel perfect: fingerprinting canvas in HTML5. In: Proc. of Web 2.0 Security and Privacy (W2SP), 2012.

[4] Mowery, K., Bogenreif, D., Yilek, S., Shacham, H.: fingerprinting information in JavaScript implementations. In: Proc. of Web 2.0 Workshop on Security and Privacy (W2SP), 2011.

[5] Nakibly, G., Shelef, G., Yudilevich, S.: Hardware fingerprinting Using HTML5. Cornell University Library, 2015.

[6] K, Takasu, T, Saito, T, Yamada, T, Ishikawa, A Survey of Hardware Features in Modern Browsers: 2015 Edition, Proc. of the 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS) 2015, 2015.

[7] Mulazzani M., Reschl P., Huber M., Leithner M., Schrittwieser S., Weippl E., Fast and reliable browser identification with JavaScript engine fingerprinting, in Proc. of Web 2.0 Workshop on Security and Privacy (W2SP), 2013.

[8] Reschl P., Mulazzani M., Huber M., Weippl E., Efficient browser identification with JavaScript engine fingerprinting, in Proc. of Annual Computer Security Applications Conference (ACSAC), 2011.

[9] scikit-learn: http://scikit-learn.org

[10] Choosing the right estimator: http://scikit-learn.org/stable/tutorial/machine_learning_map/

[11] Web Cryptography API: http://www.w3.org/TR/WebCryptoAPI/

[12] Octane 2.0: https://developers.google.com/octane/

[13] Crypto++® Library 5.6.3: http://www.cryptopp.com/

[14] T. Krovetz, W. Dai, How to get fast AES calls?: Crypto++ user group, Retrieved 2010-08-11.

[15] test262, http://test262.ecmascript.org/