An IDS for Browser Hijacking

Diogo Mónica and Carlos Ribeiro

INESC-ID/IST
Instituto Superior Técnico
Lisboa, Portugal
Email: diogo.monica@ist.utl.pt, carlos.ribeiro@ist.utl.pt

Abstract—**The steady evolution of browser tools and scripting languages has created a new, emergent threat to safe network operations: browser hijacking. In this type of attack, the user is not infected with regular malware but, while connected to a malicious or compromised website, front end languages such as javascript allow the user's browser to perform malicious activities; in fact, attackers usually operate within the scope of actions that a browser is expected to execute. Paradigmatic examples are the recent attacks on GitHub, where malicious javascript was injected into the browser of users accessing the search-giant Baidu, launching a devastating denial-of-service against a US-based company. Detecting this type of threat is particularly challenging, since the behavior of a browser is context specific. Detection can still be achieved, but to effectively hamper the effectiveness of this type of attack, users have to be empowered with appropriate detection tools, giving them the ability to autonomously detect and terminate suspicious types of browser behavior. This paper proposes such a tool. It uses information available within the browser (and is, thus, implementable as a browser extension), and it allows users to detect and terminate the suspicious types of behavior typical of hijacked browsers.**

Keywords–IDS; Browser hijacking; Malicious attack detection; User empowerment.

I. INTRODUCTION

The detection of malicious behavior that does not directly target the user's browser is largely unexplored. Since the majority of existing detection methods focus on the detection of specific attacks (exploit attempts, for example), behaviors such as having javascript code send thousands of back-to-back connections to different remote destinations is typically not considered malicious, even though it constitutes a behavior associated with several types of attacks, from browser-based denial-of-service [1][2], stealth botnet command-and-control [3] and even network reconnaissance techniques, such as network and port scanning [4]. Another example of undesired behavior that a malicious javascript might inflict on a browser is the use of the host's computational power to, for example, participate in mining crypto-currencies [5]. Unfortunately, until recently, browsers did not provide access to statistics on how a particular tab was behaving, giving the user information only on behavior of the browser as a whole. By leveraging the fact that browsers such as Google Chrome now allow the access to very granular information about the actions being taken by the browser itself, we can create tools that analyse the behavior of each tab individually, and inform the user of potentially suspicious behavior.

In this article, we will show results which indicate that, with the appropriate choice of the classification dimensions, a linear multi parameter detector is capable of flagging such attacks and, hence, of giving users the capability to disconnect from malicious sites if and when their browser is being wrongly used. In fact, empowering users with such a detector allows us to go further ahead in the road of cooperation between users and researchers. One of the problems that are yet to be solved, is the fact that even for samples of javascript that can be detected and classified as malicious by existing tools, we still have to come in contact with them in the first place. Crawling the internet seems like a losing battle; it is just not feasible to do it at a rate where a piece of malicious code can be detected before the attacker has already taken advantage of the bandwidth and processing power of several thousands of users. If users can detect the hijacking of their browsers for malicious purposes, then it will be possible to have these users' browsers automatically submit URLs where potentially malicious javascript might be running, in an anonymous fashion. This, together with the unification of the several blacklists currently in existence, could be a great defuser (and deterrent) of this kind of attack.

The vulnerabilities of front end languages such as javascript have long been recognised, and much work has been devoted to protection against the risks posed by the considerable capabilities of such languages to the user's systems (e.g., [6][7][8]). These approaches typically limit the power of javascript, either by restricting the language to an accepted subset of capabilities, or simply by monitoring the scripts degree of adherence to a designated secure framework, and dynamically modifying untrusted code to allow only operations and API calls deemed to be secure. The major drawback of such an approach lies in the fact that it effectively hampers front end capabilities for both good and evil purposes alike. The use of data driven approaches to analyse network traffic behavior and, hence, build intrusion detection systems (IDS) by anomaly detection, has also received much attention; a plethora of tools have been proposed, ranging from purely stochastic approaches, to machine learning based clustering and classification algorithms (e.g., [9][10][11][12][13][14]). These approaches are network oriented, and are typically based on network wide routing and/or protocol execution data.

Hence, most of the tools used in this paper have already been individually used and proposed in the general field of IDS, even though in different particular contexts, and with different particular objectives. To the authors knowledge, the approach proposed in this paper is, however, unique, due to the set of properties it presents: i) isolated operation: detection is achieved by individual users, without the need for any network related data; ii) No cooperation from the site hosting the script is required; iii) The set of capabilities of the scripting language is not restricted in any way; iv) the decision parameters required by the classifier are all obtainable within the user browser, thus allowing for algorithm implementation as a simple browser extension. Hence, this approach empowers users in a very effective and simple way, allowing them to autonomously detect the hijacking of their browsers for malicious purposes. As previously discussed, this will heavily contribute to safer network operation. The rest of this paper is organized as follows: Section II describes the detector, its rationale and implementation. Section III discusses the obtained experimental results. Section IV concludes the paper.

II. Detection

Even though general in scope and nature, the proposed tool is currently focused on providing an effective defense against the type of threats that use legitimate users' browsers to amplify the network capabilities of the attackers. Good examples are the javascript version of Low Orbit Ion Cannon (JS LOIC) [15], a denial-of-service (DoS) tool used by the hacker group Anonymous, and the javascript based botnet command-and-control discussed in [3]. As such, the detection problem setup will be based on the type of behavior adopted by a hijacked browser in these particular cases.

While the objective of a malicious payload that delivers a DoS tool is fundamentally different than a payload that allows attackers to do network reconnaissance, both imply a dramatic increase in the rate of new HTTP requests, and a subsequent increase in CPU usage of the browser tab that hosts the malicious code. In the particular case of the command-and-control architecture described in [3], there is an initial phase of random (or semi-random) IP scanning, where hijacked browsers are used in an attempt to disseminate malicious commands to one or more infected bots. Due to this massive scanning phase, any recruited browsers will dramatically increase the rate of new HTTP requests to distinct destination IP addresses. If we are able to reliably detect either this bot reaching behavior, or the less complex situation of a DoS based on HTTP requests flooding, the browser can automatically terminate the session with the site, or simply close the affected tab, effectively stopping the attack. Also, as previously mentioned, it will be possible at this point, to automatically submit URLs where potentially malicious javascript might be running, something which could be a great defuser of this kind of attack.

The proposed detector will thus be designed to detect this scanning and request flood behavior. No single parameter can, however, constitute a reliable indicator of this malicious behavior. An increase in HTTP requests may be the result of a legitimate action; an increase in the computational effort being consumed by the browser may easily occur in many legitimate instances (e.g., video streaming); in general, the same can be said of any single parameter/indicator. We will, thus, employ a multiparameter detector. Each one of the used dimensions will produce an indicator which, in itself, will be incapable of completely typifying the "hijacked" behavior but, taken together, they will be capable of flagging this behavior, as will be seen.

The associated detection problem is, however, not trivial. Since a false alarm will imply the disconnection from the site (or the tab) and, thus, a serious inconvenience to the user, it is important to guarantee the adequacy of the detection algorithm. Ad-hoc heuristics based on a single parameter evaluation of, for example, new HTTP requests to new IPs (mean rate, maximum rate, effective rate, accumulated number of requests, etc) are prone to fail, and are easily deceivable; multi-parameter heuristics (e.g., a linear combination of the above parameters) are more complex, they are typically highly arbitrary in the parameter weights, don't always generate an adequate final metric, and are not easily scalable, due to the many dimensions along which scaling can be independently done. The problem of multi-criterion detection will therefore be handled by a data driven mechanism, to avoid imposing arbitrary heuristic rules to the detector. In this article, the classifier will be a simple perceptron, but any equivalent linear classifier might have been used. It will be trained with different instances of "normal" and "hijacked" browser behavior. Being a linear classifier, the performance of the perceptron will depend on the two classes of behavior ("normal" and "hijacked") being, or not being, linearly separable [16]. As will be shown below, in all performed tests, the rate of correct classification was 100%, which implies that, with the chosen classification dimensions, the problem seems to be, indeed, linearly separable.

The proposed detector should be implementable as a browser extension and will, thus, be based solely on the data accessible by the browser, concerning its own behavior. It will be based on three different dimensions (all of them accessible and quantifiable within the browser itself, on a per-tab basis):

- Computational effort;
- Periodicity of new HTTP requests;
- The sequence of destination IP addresses of new HTTP requests;

The rationale behind the first indicator (computational effort) is clear. An attempt to establish an effective massive scanning strategy will necessarily correlate with an increase in the computational effort required by that browser's tab. The same is true if the browser is recruited to perform a DoS attack.

The second indicator has a more subtle rationale: independently of the computational power of the host and, therefore, of its achievable maximum rate of new HTTP requests, whenever the host is driven close to its maximum power in the massive task of flooding a single remote target or scanning the full internet address space, the new HTTP requests will become increasingly periodic, the period being dictated by the minimum cycle time achievable with the host's computational abilities; it is, thus, a dimension which, even though capable of indicating a browser hijack, is fairly insensitive to the amount of computational power available to the host computer.

The third and last indicator (sequence of destination IP addresses of new HTTP requests) is used to capture the addressing schemes typical of blind scanning strategies.

As has already been stated, none of these indicators will be capable of completely typifying the "highjacked" behavior, since periodicities will arise in legitimate video streaming, high computational loads will appear in many legitimate instances, etc; the same can be said, *mutatis mutandis*, for each one of the individual indicators. However, when considered together, they will be capable of flagging this behavior, as will be seen below.

Obtaining the indicators to feed the perceptron will require some pre-processing of the browser traffic data in each one of the considered dimensions, as will be seen. A fourth parameter (absolute number of new HTTP requests during the analysis period) will be used, not as a decision parameter, but as an enabler mask. This will be detailed below.

A. *Computational effort*

This dimension of the detection scheme evaluates the fraction of available computational effort that a particular browser tab is requiring. In a sense, this is the easiest one of the three indicators to be obtained, since we can obtain the fraction of the available computing power being consumed by a tab, directly from the browser. To account for the possibility of different profiles in the computational requirements within the analysis period, the obtained data passes through an integrator finite impulse response (FIR) filter. Hence, only the total computational power consumed in the analysis period (as a fraction of the available computational power in that period) is considered. Every second, a new measure of the fraction of computational power being consumed by the browser's tab is taken ($c(n)$, n being the time index of the sample). This sample is fed to an integrator FIR filter, which implements the composite Trapezoidal Quadrature rule, whose output p_n is the total consumed computational power over the last 5 seconds; the Trapezoidal rule was chosen due to its more acceptable behavior at sampling rates close to Nyquist, when compared with other short impulse response rules such as, for example, the Simpson rule (see, for example, [17], for details.).

The filter's gain is also adjusted, to guarantee that its output remains in the interval [0,1]. Since the interval between samples is $1\,s$, the filter's impulse response has 6 taps; the integrator filter thus becomes (noting that $0 < c(i) < 1, \ \forall\, i$):

$$p_n = \frac{1}{10}\left(c(n) + 2\sum_{i=1}^{4} c(n-i) + c(n-5)\right)$$

Since this filter is implemented as a sliding window, we have

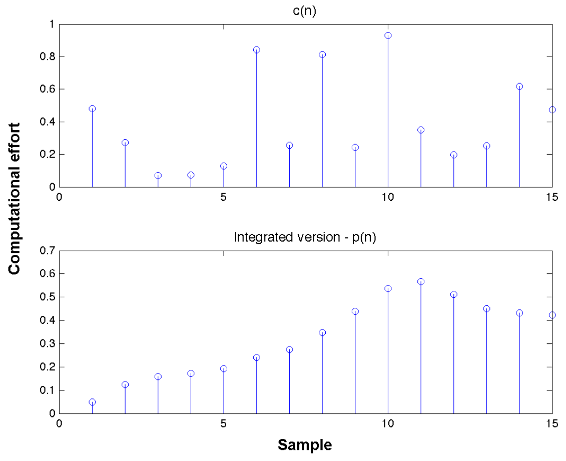

Figure 1. Computational indicator.

a new value of p_n at every second (see Figure 1), to be used as an indicator of the "computational effort" dimension, and, therefore, to be directly fed into the detector. This indicator is, therefore, valued in the $[0, 1]$ range, with higher values corresponding to higher computational loads.

B. *Periodicity of new HTTP requests*

As discussed, whenever the host is driven close to its maximum power in the single massive task of scanning the internet address space, the new HTTP requests will become increasingly periodic. The period depends on the host's computational abilities and the cycle time it can achieve, but the appearance of a periodic behavior may itself be used as an indicator. To decrease the computational complexity of the periodicity evaluation, no spectral analysis is performed. The indicator is, thus, obtained strictly via time domain processing, and relies on the analysis of the time between successive new HTTP requests.

If new connection requests appear in a purely random fashion, a Poisson arrival process is to be expected. This means that the times between successive connection requests (inter-arrival times) should be exponentially distributed (see, e.g., [18]). One useful measure of the type of use being required from the browser can, therefore, be obtained by testing the sequence of inter-arrival times, and determining if their distribution is, indeed, exponential (implying that we are facing random, non periodic, requests), or if they have some deviation from this pure theoretical random pattern. A simple Kolmogorov-Smirnoff (KS) test may be used for this purpose (e.g., [19][20]). The test proceeds as folows [19]:

As is standard in the KS test, it relies on comparing the sample cumulative distribution function with the cumulative exponential distribution function: given a sequence of N observations, one determines

$$d = max\left\{\left|S(X) - C(X)\right|\right\}, \tag{1}$$

where $S(X)$ is the sample cumulative distribution function, and $C(X)$ is a cumulative exponential distribution function with mean $\bar{\mu}$, the sample mean. If d exceeds a given threshold Ψ, one rejects the hypothesis that the observations were taken from an exponential distributed population. For sequences of 30 or more samples, and a level of significance of 0.05, the threshold Ψ may be approximated by [19]:

$$\Psi = \frac{1.06}{\sqrt{N}}, \tag{2}$$

N being the number of samples in the sequence. For lower values of N, the corresponding values of Ψ can be found in [19].

We can, therefore, easily verify if the inter-arrival times are (or are not) exponentially distributed. However, "not being exponential" is far from being a sufficient indicator that an attack is under way. As such, instead of using a binary variable to represent the exponential/not exponential nature of the inter arrival times as in the KS test, we will use the (continuous) ratio d/Ψ. Also, a complementary measure will be used: the ratio $\sqrt{\sigma^2}/\mu$ of the inter arrival times (σ^2 being their variance, and μ their mean value); when the HTTP requests become increasingly periodic, this ratio becomes increasingly smaller (since the inter-arrival times become increasingly concentrated around the mean) and, as such, it constitutes an additional measure of periodicity. To compute this ratio at each moment (the sampling period is again 1 second), we consider the inter arrival-times observed in the previous 5 seconds. The estimate for the mean arrival time is the sample mean ($\bar{\mu}$), and the estimate for the variance is obtained by the unbiased sample variance:

$$\bar{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3}$$

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{\mu})^2 \tag{4}$$

$$\tag{5}$$

Combining the two previous measures, the final indicator for periodicity of new HTTP requests becomes:

$$\text{per}_n = \frac{d\sqrt{s^2}}{3\Psi\bar{\mu}}. \tag{6}$$

The scaling factor $1/3$ is used simply to maintain a common scale to all tree used indicators.

C. The sequence of IP addresses of new HTTP requests

The behavior of a hijacked browser in what concerns the pattern of contacted IP addresses varies widely with the type of attack into which the browser is recruited. While attacks such as DoS will result in an endless repetition of a single IP (or a short list of IPs), an attack where the hijacked browser must scan the internet (as in the case of the referred stealth botnets) will generate a very high dispersion of addresses where, typically, no address is attempted more than once. To further complicate matters, the strategies for blind scanning may also differ widely, ranging from deterministic sequential scanning to purely random scanning. To address the issue with a single indicator, we used a two step procedure, applied, at each second, to the list of IP addresses of new connections attempted in the 5 s period ending at moment n ($l_n(i),\ 1 < i < M$), M being the number of new connections attempted within that period :

1) A new list L_n is obtained, by computing the absolute values of the second order divided diferences of l_n:

$$L_n = |\,\Delta(\Delta(l_n))\,|\,, \tag{7}$$

where the Δ operator is defined by:

$$\Delta(l_n) = \{l_n(i) - l_n(i-1),\ 2 < i < M\}; \tag{8}$$

2) If all elements of L_n are 0, indicator addr$_n$ is considered to be 0. Otherwise, addr$_n = \sigma_{L_n}/3\mu_{L_n}$, where μ_{L_n} is the mean value of elements in L_n, σ_{L_n} its standard deviation, and the $1/3$ factor is, again, simply a scaling factor. That is:

$$\text{addr}_n = \begin{cases} 0 & \text{if } L_n(i) = 0,\ 1 < i < M-2 \\ \frac{\sigma_{L_n}}{3\,\mu_{L_n}} & \text{otherwise} \end{cases} \tag{9}$$

This indicator thus has the following properties: i) DoS attacks to a single address will map to addr$_n = 0$ (due to the inner Δ operator); ii) Sequential scanning strategies will map to addr$_n = 0$ (due to the sequence of inner and outer Δ operators); iii) random scanning strategies will also map to low values of addr$_n$, due to the resulting high standard deviation of L_n. This means that both DOS attacks and the two most common scanning strategies (sequential and random) will generate low values of this indicator, something which makes it a powerful dimension of the detection scheme.

D. Absolute number of new HTTP requests

As will be seen, with the previous three indicators, the perceptron is capable of separating the ”normal” and ”hi-jacked” cases, for all periods with a reasonable amount of activity. However, in some periods when the browser is idle, the indicators are incapable of characterizing the activity, due to an insufficient number of new connections and, thus, the unreliabilty of the derived statistics. To account for those cases, and since the absence of traffic is a consistent indicator that no attack is under way, a threshold $\Phi = 10$ is established for the minimum number of new HTTP requests, under which it is always assumed that no attack is under way. In fact, with such low rates of probing, and the correspondingly high intervals between probes, any eventually ongoing attack (be it a DOS attack, or an attempt to scan the internet address space) would be highly ineffective, and thus inexistent as a real threat, in this context.

III. Experimental results

To train the perception performance, 50 browser sessions were logged. From these sessions, 450 non-overlapping 5 seconds periods were extracted, to be used as training set (D); of these, 150 correspond to regular browser use, 150 to a simulated DOS attack, and 150 to forced random scanning periods; 50 other periods were obtained, to be used as a test set. The three indicators x_1, x_2, and x_3 for the training 450 periods were fed to the perceptron, for supervised training; 100 iterations (epochs) were used in training, with a learning factor $\alpha = 0.1$; the perceptron weights $\mathbf{w}$ were randomly initialized. At each epoch, all samples were presented to the perceptron, sequentially, in random order. For each sample $\mathbf{x}(i) = [1, x_1(i), x_2(i), x_3(i)]$ in the training set D (the indicator vectors have been extended with a trailing 1, for mathematical convenience in the training equations below), training proceeds as follows (see, e.g., [21][22], for further details on the perceptron concept, design and training):

1) Obtain the perceptron output:

$$\begin{aligned} y(i) &= f\Big(\mathbf{w}(i)\cdot\mathbf{x}(i)\Big) = f\Big(w_0(i) + \qquad (10)\\ &+ \ w_1(i)x_1(i) + w_2(i)x_2(i) + w_3(i)x_3(i)\Big), \end{aligned}$$

where

$$f(\tau) = \begin{cases} 1 & \text{if } \tau > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

2) Update the weights:

$$\mathbf{w}(i+1) = \mathbf{w}(i)\ + \alpha(d(i) - y(i))\mathbf{x}(i), \tag{12}$$

$d(i)$ being the correct decision label (1- attack, 0 - no attack) for the i^{th} 5 second period.

After training, classification of a given observed period is done simply by computing

$$\begin{aligned} y(i) &= f\Big(\mathbf{w}(i)\cdot\mathbf{x}(i)\Big) = f\Big(w_0(i) + \qquad (13)\\ &+ \ w_1(i)x_1(i) + w_2(i)x_2(i) + w_3(i)x_3(i)\Big), \end{aligned}$$

with $\mathbf{w}$ being the final set of weights that resulted from the training phase, and $\mathbf{x}(i) = [1, x_1(i), x_2(i), x_3(i)]$ the augmented indicator vector for that period.

When the 450 training periods were run through the trained perceptron, no errors in classification were observed, which means that the problem, as mapped by these three indicators, is indeed linearly separable, within the training set. For the test set (constituted by 50 5-seconds period, as discussed above), classification was also 100% successful, with no misclassifications. Even though the number of examples used in this paper is somewhat limited, and no real life attacks have been used, the obtained results seem to indicate that the proposed indicators may, indeed, be capable of transforming the browser highjacking detection problem into a linearly separable one, thus addressable by simple linear classifiers, implementable by simple, lighweight, browser extensions.

IV. Conclusion

Our results show that, with the appropriate choice of indicators, it seems to be possible to create a linearly separable setup, amenable to the detection of browser hijacking by malicious sites with a simple linear detector. This conclusion must still be validated with bigger, real life, datasets. Detection is accomplished using only variables which the browser provides access to, and it can be done with a per-tab granularity. Two main types of attack are thus defused: attacks that utilize honest user's browsers as a platform to launch denial-of-service, and attacks which imply mass IP address scanning (both sequential and random). In particular, this detector allows users to become a vital part in defusing a particularly dangerous type of stealth botnet, by detecting that their browser is being used as part of the botnet command-and-control structure. Finally, our solution also paves the way for automatic submission of URLs where potentially malicious javascript might be running, something which can be a great defuser and deterrent for future attacks.

References

[1] J. Temperton, 2015, [Accessed 11 July 2015]. [Online]. Available: http://www.wired.co.uk/news/archive/2015-04/10/china-great-cannon-github-hack

[2] K. Higgins, 2013, [Accessed 11 July 2015]. [Online]. Available: http://www.darkreading.com/attacks-breaches/ddos-attack-used-headless-browsers-in-150-hour-siege/d/d-id/1140696

[3] D. Oliveira and C. Ribeiro, "Leveraging honest users: Stealth command-and-control of botnets," in Proceedings of the 7^{th} USENIX Workshop on Offensive Technologies (WOOT '13), Washington, D.C., August 2013.

[4] L. Kuppan, 2010, [Accessed 11 July 2015]. [Online]. Available: http://www.andlabs.org/tools/jsrecon.html

[5] R. King, 2013, [Accessed 11 July 2015]. [Online]. Available: http://qz.com/154877/by-reading-this-page-you-are-mining-bitcoins/

[6] P. Phung, M. Monshizadeh, M. Sridhar, K. W. Hamlen, and V. Venkatakrishnan, "Between worlds: Securing mixed javascript/actionscript multi-party web content." in IEEE Trans. Dependable and Secure Computing, 2014.

[7] P. Phung, D. sands, and A. Chudnov, "Lightweight self-protecting javascript," in Proceedings of ASIACCS09, March 2009.

[8] Y. Dachuan, A. Chander, N. Islam, and I. Serikov, "Javascript instrumentation for browser security," in Martin Hofmann 0001 Matthias Felleisen, ed., POPL, ACM, 2007, pp. 237–249.

[9] H. Kayacik, A. Zincir-Heywood, and M. Heywood, "On the capability of an som based intrusion detection system," in Proc. Int. Joint Conf. Neural Networks, vol. 3, July 2003, pp. 1808–1813.

[10] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, and M. Embrechts, "Network-based intrusion detection using neural networks," in Proc. Artif. Neural Netw. Eng., vol. 12, November 2002, pp. 579–584.

[11] P. Kabiri and A. Ghorbani, "Research on intrusion detection and response: A survey." in International Journal of Network Security, vol. 1(2), 2005, pp. 84–102.

[12] J. Cabrera, B. Ravichandran, and R. Mehra, "Statistical traffic modeling for network intrusion detection," in Proc. Modeling, Anal. Simul. Comput. Telecommun. Syst., 2000, pp. 466–473.

[13] D. Denning, "An intrusion detection model," in IEEE Trans. Softw. Eng., vol. SE-13, N.2, February 1987, pp. 222–232.

[14] S. Jiang, X. Song, H. Wang, J. Han, and Q. Li, "A clustering-based method for unsupervised intrusion detections," in Pattern Recognition Letters, Elsevier, vol. 27, 2006, pp. 802–210.

[15] P. Shankdhar, 2013, [Accessed 11 July 2015]. [Online]. Available: http://resources.infosecinstitute.com/dos-attacks-free-dos-attacking-tools/

[16] F. Rosenblatt, "The perceptron–a perceiving and recognizing automaton," in Report 85-460-1, Cornell Aeronautical Laboratory, Jan 1957.

[17] R. Hamming, Digital filters, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1983.

[18] A. Papoulis, Probability, Random Variables and Stochastic Processes, 2nd ed. New York, USA: McGraw-Hill, 1984.

[19] H. W. Lilliefors, "On the kolmogorov-smirnov test for the exponential distribution with mean unknown," in Journal of the American Statistical Association, vol. 64, March 1969, pp. 387–389.

[20] V. Seshadri, M. Csorgo, and M. A. Stephens, "Tests for the exponential distribution using kolmogorov-type statistics," in Journal of the Royal Statistical Society. Series B (Methodological), vol. 31, 1969, pp. 499–509.

[21] C. M. Bishop, Neural Networks for Paitern Recognition. Oxford: Clarendon Press, 1995.

[22] R. O. Duda and e. a. P. E. Halt, Pattern Classification. Wiley-lnterscience, John Wiley and Sons, 2001.